Partner datenverarbeitender Services

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. rer. nat.
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
Humboldt-Universität zu Berlin

von
Dipl.-Inf. Christoph Wagner

Präsident der Humboldt-Universität zu Berlin:
Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät:
Prof. Dr. Elmar Kulke

Gutachter:
1. Prof. Dr. Wolfgang Reisig
2. Prof. Dr. Karsten Wolf

eingereicht am: 3. Juni 2014
Tag der Verteidigung: 9. Dezember 2014

Kurzfassung

Diese Arbeit untersucht den Einfluss von *Daten* auf das Verhalten und die *Korrektheit* eines verteilten Systems.

Ein verteiltes System besteht aus mehreren *Services*. Ein Service ist eine selbständige, plattformunabhängige Einheit, die anderen Services eine bestimmte Funktionalität über eine wohldefinierte Schnittstelle zur Verfügung stellt. Die Services kommunizieren untereinander durch Senden und Empfangen von Nachrichten. Eine Nachricht kann Daten aus einem potentiell unendlich großen Wertebereich enthalten.

Services werden mit dem Ziel entworfen, dass sie eine bestimmte *Aufgabe* erfüllen. Die Aufgabe kann z. B. darin bestehen, einen Geschäftsprozess erfolgreich abzuschließen. Ein Service ist stets eingebunden in ein verteiltes System, das aus mehreren Services besteht. Zum Erfüllen der Aufgabe sind die Services aufeinander angewiesen. Kein Service allein kann die Aufgabe erfüllen, sondern muss dazu mit mindestens einem anderen Service interagieren.

Jeder Service wird im allgemeinen von einer anderen Partei zur Verfügung gestellt. Diese Parteien finden sich zum Lösen einer gemeinsamen Aufgabe zusammen und treffen dabei häufig zum ersten Mal aufeinander. Um das gemeinsame Ziel zu erreichen, müssen die Parteien zusammenarbeiten. Jeder Partei sind die internen Abläufe der Services der anderen Parteien in der Regel unbekannt. Trotz der wechselseitigen Unkenntnis der Parteien untereinander muss gewährleistet sein, dass die Services korrekt miteinander interagieren, so dass stets das gemeinsame Ziel erreicht werden kann.

In dieser Arbeit betrachten wir die Interaktion von jeweils genau zwei Services miteinander. Damit zwei Services erfolgreich miteinander zusammenarbeiten können, muss ihr Verhalten miteinander *kompatibel* sein. Miteinander kompatible Services nennen wir *Partner*.

Ziel dieser Arbeit ist es, zu entscheiden, ob ein Service mindestens einen Partner hat. Aufgrund der Daten kann der Zustandsraum eines Service sehr groß oder sogar unendlich groß werden. Daher ist es nicht trivial, zu entscheiden, ob ein Service einen Partner hat. Wir untersuchen zwei Klassen von Services mit unendlich vielen Zuständen. Für diese Klassen stellen wir Algorithmen vor, welche zu einem gegebenen Service einen Partner synthetisieren, falls ein solcher existiert. Auf diese Weise entscheiden wir die Existenz eines Partners. Ferner untersuchen wir, wie viel Speicherplatz ein Partner eines Services mindestens benötigt.

Abstract

This thesis studies the influence of *data* on the behavior and the *correctness* of a distributed system.

A distributed system consists of several *services*. A service is a self-contained, platform-independent entity which provides a certain functionality to other services via a well-defined interface. The services communicate with each other by sending and receiving messages. A message may contain *data* from a potentially infinite domain.

Services are designed with the goal in mind that they can fulfill a specific *task*. This task may, for example, involve to successfully completete a business process. A service is always embedded into a distributed system that consists of several services. The services rely on each other to fulfill the task. No single service can accomplish the task on its own. A service has to interact with at least one other service in order to fulfill the task.

Each service is provided by a different party. These parties come together to solve a common goal. Thereby, the parties often meet each other for the first time. In order to achieve their common goal, all parties have to cooperate with each other. In general, no party knows the internal processes of the services of the other parties. Despite the fact that the parties do not know each other, correct interaction between the services must be guaranteed, so that their common goal can always be achieved.

In this thesis, we consider the interaction of exactly two services. In order to successfully cooperate with each other, the behaviour of the two services must be *compatible*. We call compatible services *partners*.

The goal of this thesis is to decide whether a given service has at least one partner. Due to the data, the state space of a service may be very large or even infinite. Therefore, deciding the existence of a partner is a non-trivial problem.

We investigate two classes of services with infinitely many states. For these classes, we present algorithms that synthesize a partner of a service, if it exists. This allows us to decide the existence of a partner. Furthermore, we investigate the minimum amount of memory a partner of a service needs.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I.

Einführung und Grundlagen

1. Einführung und Ziele der Arbeit

Arbeitsabläufe, an denen mehrere Organisationen beteiligt sind, sind im allgemeinen verteilt organisiert. Die beteiligten Parteien befinden sich an verschiedenen Orten und tauschen Informationen und reale Güter miteinander aus. An diesem Austausch sind häufig rechnergestützte Informationssysteme beteiligt. Der Austausch muss koordiniert ablaufen, damit die Organisationen zusammenarbeiten können.

Diese Arbeit wurde aus Mitteln des Graduiertenkollegs *METRIK - Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informationssysteme im Katastrophenmanagement* finanziert. Daher soll hier ein Szenario aus dem Katastrophenmanagement als ein Beispiel für ein verteiltes, von mehreren Organisationen gebildetes System dienen.

Nach dem Eintreten einer Naturkatastrophe - etwa einer Flut - kommen viele Parteien und Organisationen zusammen, um im Katastrophengebiet Hilfe zu leisten. Diese Parteien sind meist heterogen zusammengesetzt. Diese setzen sich zusammen aus staatlichen Organisationen wie Polizei, Feuerwehr, dem Militär sowie Ärzten, dem Technischen Hilfswerk, Freiwilligen und anderen lokalen Helfern.

Alle diese Parteien müssen zusammenarbeiten, um im Einsatzgebiet erfolgreich Hilfe leisten zu können. Dämme müssen errichtet werden, Verletzte müssen versorgt und Erste Hilfe muss geleistet werden. Dabei sind die beteiligten Parteien aufeinander angewiesen. Medizinische Ausrüstung muss verteilt werden, Sandsäcke und andere Ausrüstungsgegenstände müssen verteilt werden.

Keine der Organisationen kann ihr Ziel allein erreichen. Feuerwehr und Militär sind beim Errichten eines Dammes auf freiwillige Helfer angewiesen. Weiterhin sind diese auf die Ausrüstung und technische Unterstützung des Technischen Hilfswerks angewiesen.

Die beteiligten Organisationen unterscheiden sich hinsichtlich ihrer Organisationsstrukturen. Ihre internen Abläufe sind in der Regel nicht aufeinander abgestimmt. Viele der Organisationen treffen zum ersten Mal aufeinander.

Es muss gewährleistet sein, dass alle Organisationen zusammen ihr gemeinsames Ziel - etwa die Errichtung eines Dammes - erreichen können.

Die an diesem verteilten System beteiligten Parteien können wir als *Services* darstellen. Ein Service ist eine selbständige Einheit, welche einen Dienst anbietet, der von anderen Services genutzt werden kann. Services kommunizieren untereinander durch Senden und Empfangen von Nachrichten. Die in dieser Arbeit betrachteten Services sind zustandsbehaftet.

Ziel der Arbeit ist es, die *Korrektheit* der Interaktion der Services zu analysieren. In dieser Arbeit betrachten wir die Interaktion von jeweils genau zwei Services miteinander. Damit zwei Services erfolgreich miteinander zusammenarbeiten können, muss ihr Verhalten miteinander *kompatibel* sein. Miteinander kompatible Services nennen wir *Partner*.

Ein Service, der keinen Partner hat, ist nicht sinnvoll nutzbar. In dieser Arbeit charakterisieren wir die Existenz von Partnern mit bestimmten Eigenschaften. Ein Schwerpunkt dieser Arbeit liegt in der Betrachtung des Einflusses von *Daten* auf die Kompatibilität von Services.

Um formal definieren zu können, wann ein verteiltes System korrekt ist, benötigen wir ein *formales Modell*. Dieses formale Modell abstrahiert in geeigneter Form von der Wirklichkeit und bildet nur die relevanten Eigenschaften eines Service ab. Mit Hilfe eines formalen Modells kann ein Service und sein Verhalten durch formale Methoden analysiert werden.

In dieser Arbeit konzentrieren wir uns auf *funktionale Eigenschaften*. Nicht-funktionale Eigenschaften wie die Zeit oder die Kosten zur Ausführung eines Service werden nicht betrachtet.

Es gibt viele verschiedene Formalismen zur Beschreibung von Services. Formal beschreiben wir einen Service in dieser Arbeit durch eine Erweiterung *endlicher Automaten*. Wir erweitern den endlichen Automaten so, dass mit diesem die Verarbeitung von *Daten* durch den Service beschrieben werden kann.

Daneben werden in der Literatur weitere Formalismen verwendet.

Ein wichtiger Formalismus sind *Offene Netze* [6, 57]. Diese werden aufgrund ihres engen Bezuges zu Workflows auch als *open workflow nets* bezeichnet. Einige der in dieser Arbeit verwendeten Konzepte wurden ursprünglich für Petrinetze formuliert. Offene Netze erweitern *Petrinetze* [78] um *Interfaceplätze* zum Senden und empfangen von Nachrichten. Beschränkte Petrinetze können in äquivalente endliche Automaten umgewandelt werden.

Arbeitsabläufe in Unternehmen (*Workflows*) und Web Services werden häufig mit Sprachen wie der Business Process Modeling Language (BPMN) und der Business Process Execution Language (BPEL) beschrieben. Lohmann definiert in [52] eine Semantik für BPEL Prozesse und eine Transformation dieser Prozesse in Petrinetze. BPEL Prozesse sind oftmals Teil eines offenen Systems und können daher in offene Netze übersetzt werden.

In der Literatur werden Services häufig mit Prozessalgebren [10] beschrieben. Dies betrifft insbesondere Web-Services [80]. Es gibt viele verschiedene Varianten von Prozessalgebren, Zu den bekanntesten zählen Milners klassische CCS [64], CSP [35] oder LOTOS [12]. LOTOS erlaubt insbesondere die Spezifikation von Daten.

Oclets [20, 22] erlauben eine szenariobasierte Beschreibung des Verhaltens von Services. Aus mit Oclets spezifizierten verteilten Abläufen können Services mit dem spezifizierten Verhalten abgeleitet werden.

Damit der Einfluss von Daten auf das Verhalten eines Services analysiert werden kann, muss im verwendeten formalen Modell spezifiziert werden, wie Daten durch den Service manipuliert werden.

In vielen Formalismen werden Daten nur in stark abstrahierter Form behandelt. Das Verhalten eines Service kann damit häufig nur grob wiedergegeben werden. Einige für die korrekte Interaktion von Services relevante Eigenschaften des Verhaltens können damit möglicherweise nicht wiedergegeben werden. Wir benötigen ein Servicemodell, welches Daten in expliziter Form berücksichtigt und in dem für die Analyse der Korrektheit

relevanten Aspekte ausgedrückt werden können.

Es gibt verschiedene Erweiterungen von Petrinetzen um Daten, welche unter dem Oberbegriff *High-Level Petrinetz* zusammengefasst werden. Dazu zählen *gefärbte Petrinetze* [38] algebraische Petrinetze [92], und Petrinetzschemata [79]. Daten werden in High-Level Petrinetzen durch farbige Marken dargestellt.

CPN Tools [77] sind ein bekanntes Werkzeug zum Modellieren gefärbter Petrinetze.

Zwei Services betrachten wir formal als kompatibel, wenn beide zusammen stets ihren jeweiligen Endzustand erreichen können.

Damit können wir die in dieser Arbeit betrachteten Hauptprobleme formal formulieren:

1. *Entscheidungsproblem*: Sind zwei Services füreinander Partner?
2. *Bedienbarkeit*: Hat ein Service einen Partner?
3. *Synthese*: Wie konstruiert man zu einem Service einen Partner, falls ein solcher existiert?

Diese Probleme sind gelöst für ein auf endlichen Automaten basierendes Servicemodell, welches Daten nicht explizit berücksichtigt. Für Modelle von Services, die Daten berücksichtigen sind diese Probleme noch offen.

Modelle von Systemen, die Daten berücksichtigen sind in der Regel wesentlich schwerer zu analysieren als Modelle von Systemen, die dies nicht tun. Daten können sehr viele verschiedene Ausprägungen haben. Der Zustandsraum eines Service kann durch die Einführung von Daten sehr groß werden. Dadurch wird die Berechnung des Zustandsraumes mit rechnergestützten Mitteln erschwert. Dieses Problem ist in der Literatur als *Zustandsraumexplosion* bekannt.

Mit Hilfe von Binary Decision Diagrams (BDDs) [13] können Zustandsräume mit Milliarden von Zuständen kompakt repräsentiert werden. Symbolisches Modelchecking verwendet BDDs zum Verifizieren von Systemen. Couvreur et al. benutzen *Data Decisision Diagrams*, eine Erweiterung von BDDs, um Petrinetze zu analysieren [18]. Zustandsräume von Services mit unendlich vielen Zuständen sind nicht mit BDDs darstellbar.

In dieser Arbeit betrachten wir Services mit unendlich vielen Zuständen. Für Services mit unendlich vielen Zuständen stellt sich zunächst die grundlegende Frage nach der Entscheidbarkeit der betrachteten Probleme. Um die Entscheidbarkeit der untersuchten Probleme zu garantieren, ist es notwendig, die Ausdrucksfähigkeit der betrachteten Services einzuschränken.

Wir untersuchen in dieser Arbeit zwei Klassen von Services mit unendlich vielen Zuständen. Für diese stellen wir Algorithmen vor, um die oben definierten Probleme zu lösen. Ferner untersuchen wir, wie viel Speicherplatz bzw. wie viele Variablen ein Partner eines Services mindestens benötigt.

1.1. Gliederung und Beiträge der Arbeit

Die Arbeit gliedert sich in drei aufeinander aufbauenden Teile.

Teil 1 Der erste Teil führt das in dieser Arbeit verwendete formale Modell zur Darstellung von Services ein. *Serviceautomaten* erweitern endliche Automaten um Variablen, so dass Services mit unendlich vielen Zuständen dargestellt werden können. Serviceautomaten ermöglichen es, die Verarbeitung von Daten durch den Service mit Hilfe von Funktionen und Bedingungen zu spezifizieren. Die konkrete Repräsentation der Funktionen und Bedingungen lassen wir an dieser Stelle bewusst offen. Diese wird im Verlauf der Arbeit für jede Klasse von Serviceautomaten passend gewählt.

Es werden elementare Operationen wie die Komposition von Services, das Verschmelzen von Knoten und das Eliminieren von Variablen definiert.

Obwohl Services eines verteilten Systems in der Regel asynchron kommunizieren, verwenden wir bei der Komposition von Services aus formalen Gründen ein synchrones Kommunikationsmodell. Wir führen danach asynchrone Kommunikation auf synchrone Komposition zurück.

Teil 2 Teil 2 führt das Konzept des *Partners* eines Serviceautomaten ein und betrachtet zunächst grundlegende Fragestellungen zur Existenz von Partnern. Es werden verschiedene *Kompatibilitätskriterien* vorgestellt, welche definieren, wann zwei Services Partner füreinander sind. Wir verwenden in dieser Arbeit durchgängig das Kompatibilitätskriterium *schwache Terminierung*.

Ein Service heißt *bedienbar*, wenn er mindestens einen Partner hat. Wir untersuchen, wann ein Service bedienbar ist und welche Art Partner er hat. Wir zeigen, dass im allgemeinen unentscheidbar ist, ob zwei Serviceautomaten Partner sind und dass Bedienbarkeit unentscheidbar ist. Diese Beobachtung motiviert die Definition von *Identitätsautomaten*, einer besonders einfachen Klasse von Serviceautomaten, die nur ein Minimum an Ausdrucksmitteln verwendet.

Wir zeigen, dass Variablen eines Partners unter bestimmten Umständen überflüssig sind und eliminiert werden können. Wir zeigen außerdem, dass innerhalb der Klasse der Identitätsautomaten ein Partner eines zyklischen Service im allgemeinen unendlich viele Variablen hat. Aufgrund dieser Beobachtung schränken wir unsere weiteren Betrachtungen auf azyklische Serviceautomaten ein.

Teil 3 Der dritte Teil stellt Algorithmen zum Entscheiden der Bedienbarkeit von Serviceautomaten vor. Wir entscheiden die Bedienbarkeit eines Serviceautomaten konstruktiv durch die Synthese eines Partners. Wir stellen Algorithmen für verschiedene Klassen azyklischer Serviceautomaten vor.

- Zunächst stellen wir einen Algorithmus zum Synthetisieren von Partnern azyklischer Serviceautomaten mit endlich vielen Zuständen vor. Die Korrektheit der anderen Algorithmen können wir auf die Korrektheit dieses Algorithmus zurückführen.

- Für azyklische *Formelautomaten*, eine Klasse von Serviceautomaten, deren Prädikate in einer entscheidbaren Theorie des Prädikatenkalküls erster Stufe formuliert sind, stellen wir einen Algorithmus zur *Synthese* eines Partners vor. Für Teilklassen

von Formelautomaten, deren Prädikate in einer entscheidbaren Theorie (wie etwa *Presburger-Arithmetik*) formuliert sind, kann mit diesem Algorithmus effektiv ein Partner synthetisiert werden.

Wir demonstrieren beispielhaft, wie Partnersynthese zur Synthese eines *Adapters* verwendet werden kann.

- Für azyklische Identitätsautomaten (einem Spezialfall von Formelautomaten) stellen wir einen eigenen Algorithmus zur Synthese eines Partners vor, der ohne die Verwendung prädikatenlogischer Formeln auskommt. Die Motivation, einen eigenen Algorithmus für diese Teilklasse von Formelautomaten vorzustellen, speist sich aus der Hoffnung, diesen so erweitern zu können, dass auch Partner zyklischer Serviceautomaten synthetisiert werden können. Wir zeigen, dass Knoten eines Partners eines Identitätsautomaten unter bestimmten Bedingungen miteinander verschmolzen werden können. Dies stellt die Möglichkeit in Aussicht, einen zyklischen Partner durch Faltung aus einem azyklischen zu erzeugen. Das Problem, Partner für zyklische Serviceautomaten zu synthetisieren, bleibt in dieser Arbeit ungelöst.

Als Nebenresultat der oben genannten Synthesealgorithmen erhalten wir Verfahren, um zu entscheiden, ob zwei gegebene Services der beiden genannten Klassen Partner füreinander sind. Ob die Services azyklisch sind, spielt dabei keine Rolle.

2. Ein formales Modell für Services

In diesem Abschnitt führen wir die formalen Grundlagen und Konzepte ein, die in dieser Arbeit verwendet werden. Insbesondere stellen wir *Serviceautomaten* als formales Modell für Services vor.

2.1. Grundbegriffe und Notation

Wir führen zunächst formale Grundbegriffe und Notationen ein.

Die *Kardinalität* einer Menge M notieren wir mit $|M|$. Wir schreiben $\mathbb{N}$ für die Menge der natürlichen Zahlen einschließlich 0 und $\mathbb{Z}$ für die Menge der ganzen Zahlen. Eine *Multimenge* über M ist eine Funktion $M \to \mathbb{N}$. Die Menge der Multimengen über M notieren wir mit $\text{bags}(M)$. Multimengen notieren wir in eckigen Klammern. Wir schreiben z. B. $[a, b, b]$ für die Multimenge über $\{a, b\}$, die einmal das Element a und zweimal das Element b enthält. Die *symmetrische Differenz* zweier Mengen A und B ist $A \triangle B \underset{Def}{=} (A \setminus B) \cup (B \setminus A)$.

Die *Komposition* zweier Funktionen $f : A \to B$, $g : B \to C$ ist die Funktion $f \circ g : A \to C$ mit $(f \circ g)(x) = g(f(x))$ für jedes $x \in A$. Eine *Permutation* einer Menge M ist eine Bijektion $\sigma : M \to M$.

Eine *Äquivalenzrelation* über einer Menge M ist eine reflexive, transitive und symmetrische Relation $R \subseteq M \times M$. Die *Äquivalenzklasse* eines Elements $m \in M$ ist $[m]_R \underset{Def}{=} \{m' \in M | m \, R \, m'\}$. Wir schreiben kurz $[m]$, falls R vom Kontext klar ist und keine Verwechselung mit Multimengen möglich ist. Die *Zerlegung* einer Menge M nach R ist die Menge der Klassen $M \backslash_R \underset{Def}{=} \{[m]_R | m \in M\}$.

Wir fixieren im folgenden eine Menge $\mathcal{U}$ von Grundelementen, das *Universum*, und die Menge $\mathcal{X}$ der *Variablen*.

Sei $X \subseteq \mathcal{X}$ eine Menge von Variablen. Dann heißt eine Funktion $\beta : X \to \mathcal{U}$ *Belegung* (über X). Eine Menge P von Belegungen über X heißt *Prädikat* (über X).

Die *Einschränkung* einer Belegung $\beta : X \to \mathcal{U}$ auf eine Variablenmenge $X' \subseteq X$ ist die Belegung $\beta_{|X'} : X' \to \mathcal{U}$ mit $\beta_{|X'}(x) = \beta(x)$ für jedes $x \in X'$. Die Einschränkung eines Prädikates P auf X' ist die Menge der Einschränkungen der Belegungen in P auf X'.

$\beta' : X' \to \mathcal{U}$ heißt *Erweiterung* einer Belegung $\beta : X \to \mathcal{U}$ auf $X' \supseteq X$, wenn $\beta(x) = \beta'(x)$ für $x \in X'$. Die Erweiterung eines Prädikates P auf X' ist die Menge der Erweiterungen der Belegungen in P auf X'.

Ein Prädikat wird formal üblicherweise durch eine prädikatenlogische Formel dargestellt. Prädikatenlogische Formeln führen wir formal in Abschnitt 11.1 ein. Konjunktion, Disjunktion und Negation einer Formel entsprechen Durchschnitt, Vereinigung und

Komplement von Prädikaten. Die Existenzquantisierung einer Variable einer Formel entspricht der Einschränkung des Prädikates auf die Variablenmenge ohne die quantisierte Variable.

Ein Prädikat kann alternativ auch als eine Äquivalenzklasse von Belegungen, spezifiziert durch einen Repräsentanten, dargestellt werden. Eine äquivalenzklassenbasierte Darstellung von Belegungsmengen verwenden wir in Abschnitt 13.1.

Auf die konkrete Form der Darstellung eines Prädikates legen wir uns in dieser Arbeit nur dann fest, wenn sie formal eine Rolle spielt. Zur Darstellung der Serviceautomaten wählen wir in allen Beispielen eine informelle Notation.

2.2. Serviceautomat

In diesem Abschnitt führen wir *Serviceautomaten* als formale Repräsentation von Services ein. Services kommunizieren durch den Austausch von *Nachrichten* über ihre Nachrichtenkanäle. Die Nachrichtenkanäle bilden das *Interface* eines Serviceautomaten. Eine Nachricht kann *Daten* enthalten, welche wir als Tupel von Werten notieren.

Serviceautomaten kommunizieren in unserem Modell grundsätzlich *synchron* miteinander. Um asynchrone Kommunikation in unserem Servicemodell darzustellen, können wir Serviceautomaten mit einem *Nachrichtenpuffer* komponieren (siehe Abschnitt 2.3.1). Obwohl im synchronen Kommunikationsmodell nicht zwischen dem Senden und dem Empfangen einer Nachricht unterschieden wird, werden wir die Kommunikation zwischen Serviceautomaten gelegentlich als Austausch von Nachrichten beschreiben. Dies macht die Funktionsweise vieler Serviceautomaten anschaulicher und hilft, den Informationsfluss zwischen Serviceautomaten verständlicher darzustellen.

Abbildung 2.1 zeigt einen Serviceautomaten. Ein Serviceautomat A besteht aus einer Menge von *Knoten* Q, einer Menge von *Nachrichtenkanälen* C, einer Menge gerichteter *Kanten* E, einem *Anfangsknoten* q_0 und einer Menge von *Endknoten* Ω.

Der Anfangsknoten wird durch einen eingehenden Pfeil gekennzeichnet. Einen Endknoten stellen wir mit doppelter Umrandung dar.

Jedem Knoten und jedem Kanal wird von einer Funktion *var* jeweils eine endliche (ggf. leere) Menge von Variablen zugeordnet. Jede Variable eines Knotens repräsentiert einen *Speicherplatz* des Service. Eine Variable eines Kanals korrespondiert zu einem Datenelement, das in einer Nachricht über den Kanal kommuniziert wird.

Die Beschriftung einer Kante eines Serviceautomaten besteht aus zwei Teilen. Eine Kante ist zum einen beschriftet entweder mit einem Kanal oder dem speziellen Symbol τ. Das Symbol τ bezeichnet eine interne, nicht kommunizierende Aktion des Serviceautomaten. Hinter dem Kanal notieren wir die ihm zugeordneten Variablen.

Zum anderen steht an jeder Kante ein Prädikat, der sogenannte *Guard* der Kante. Der Guard spezifiziert gleichzeitig die *Vorbedingung*, die erfüllt sein muss, damit die Kante ausgeführt werden kann, und die *Nachbedingung*, die nach dem Ausführen der Kante gilt und gegebenenfalls den Inhalt der Nachricht, welche die Kante auf ihrem Kanal kommuniziert.

Die Variablen des Zielknotens einer Kante werden im Guard durch einen hochge-

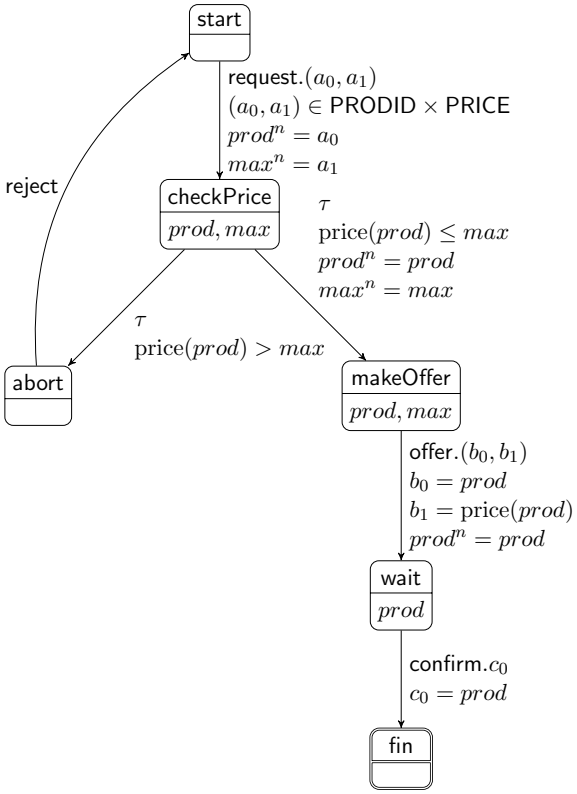


Abbildung 2.1.: Serviceautomat Shop

stellten Index new (Kurzschreibweise auch n) gekennzeichnet, um sie von gleichnamigen Variablen des Quellknotens zu unterscheiden. Zuweisungen einer Variablen an sich selbst der Art $x := x+1$ können auf diese Weise als Prädikat der Form $x^{\text{new}} = x+1$ dargestellt werden.

Andere vergleichbare Modelle wie Extended Finite Automata (EFA) [84] bestimmen die Belegung der Variablen nach Ausführen einer Kante durch eine unabhängig vom Guard spezifizierte Funktion. Für die Operationen, die wir auf unsere Serviceautomaten anwenden werden, ist es jedoch vorteilhaft, den Effekt einer Kante durch ein einziges Prädikat auszudrücken.

Beispiel Der Serviceautomat Shop in Abb. 2.1 zeigt ein Geschäft mit Kanälen request, reject, offer und confirm. Der Kanal request hat die zwei Variablen a_0 und a_1. Der Kanal reject hat eine leere Variablenmenge.

Der Serviceautomat Shop nimmt von einem Kunden eine Anfrage entgegen auf dem Kanal request. Die Anfrage besteht aus dem angefragten Produkt (in a_0) und einer Obergrenze (in a_1) für den Preis, den der Kunde bereit ist, für dieses Produkt zu zahlen. Die beiden Daten werden in den Variablen *prod* und *max* gespeichert.

Die Variable *prod* nimmt Werte aus einer Menge PRODID von Bestellnummern von Produkten an und *max* Werte aus einer Menge von Preisen PRICE.

Wenn das Produkt teurer ist als die angegebene Preisobergrenze des Kunden, wird die Anfrage des Kunden abgelehnt (reject). Der Serviceautomat geht danach zurück in seinen Anfangszustand und wartet auf weitere Anfragen. Andernfalls schickt er dem Kunden ein Angebot, in dem er dem Kunden den Preis des Produktes mitteilt (offer). Nach Eingang der Bestätigung (confirm) durch den Kunden schließt der Serviceautomat das Geschäft ab und geht in seinen Endknoten.

Der Typ einer Variablen wird in unserem formalen Modell nicht explizit spezifiziert. Jede Variable kann prinzipiell mit jedem beliebigen Wert aus dem Universum $\mathcal{U}$ belegt werden. Der Bereich der Werte, die eine Variable tatsächlich annehmen kann, kann bei Bedarf durch die Guards eingeschränkt werden. In technischen Beispielen, in denen wir keine expliziten Einschränkungen angeben, nehmen wir stets an, dass $\mathcal{U}$ die Menge der ganzen Zahlen ist.

Wir definieren nun Serviceautomaten formal. Die Funktion varnew ordne dabei jeder Variablen x eines Knotens des Serviceautomaten jeweils eine neue, d. h. in keinem anderen Knoten benutzte Variable x^{new} zu.

Definition 2.1 (Serviceautomat) *Ein Serviceautomat* $A = (Q, C, var, q_0, E, \Omega)$ *besteht aus*

- *einer (ggf. unendlich großen) Menge von* Knoten Q*.*
- *einer Menge* C *von* Nachrichtenkanälen
- *einer Funktion* $var : Q \cup C \cup \{\tau\} \to \mathcal{X}$*, die jedem Knoten und jedem Kanal eine endliche Menge von Variablen zuordnet und* q_0 *und* τ *die leere Menge zuordnet.*

- *einem* Anfangsknoten $q_0 \in Q$ *mit* $var(q_0) = \emptyset$
- *einer Menge von* Kanten E
- *einer Menge* $\Omega \subseteq Q$ *von* Endknoten.

Die Variablenmengen der Kanäle sind jeweils zueinander disjunkt und disjunkt zu den Variablen der Knoten.
Eine Kante *ist ein Tupel* $e = (q, c, G, q')$*, wobei*

- $q \in Q$ *der* Quellknoten *e,*
- $q' \in Q$ *der* Zielknoten *von e,*
- $c \in C \cup \{\tau\}$ *ein Kanal oder das Symbol* τ
- G *ein Prädikat über der Variablenmenge* $var(q) \cup var(c) \cup \text{varnew}(var(q'))$

G *nennen wir den* Guard *von e. Wir schreiben kurz* $var(e)$ *für die Variablenmenge des Guards von e.*

Die Variablen der einzelnen Knoten eines Serviceautomaten müssen im Gegensatz zu den Variablen der Kanäle nicht disjunkt voneinander sein. Endknoten können im Gegensatz zum Anfangsknoten Variablen haben.
Ein *Pfad* ist eine Folge von Kanten eines Serviceautomaten.

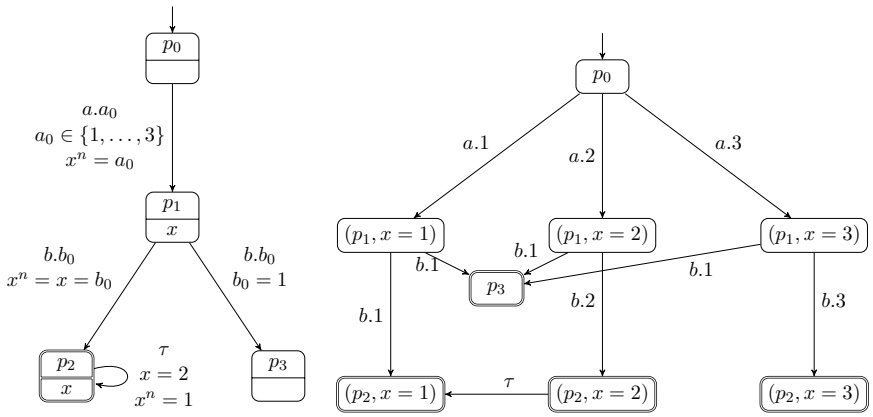
Definition 2.2 (Pfad) *Sei A ein Serviceautomat. Ein* Pfad *(der Länge n) von einem Knoten q von A zu einem Knoten* q' *ist eine Folge von Kanten* $e_1 \dots e_n$*, so dass es Knoten* $q_0, \dots, q_n$ *gibt mit* $q_0 = q$*,* $q_n = q'$ *und* $e_i = (q_i, c_i, G_i, q_{i+1})$ *für* $0 \leq i \leq n-1$*.*

Wir definieren nun das *Transitionssystem* eines Serviceautomaten. Abb. 2.2 zeigt einen Serviceautomaten und sein Transitionssystem.

Definition 2.3 (Transitionssystem) *Sei* Σ *ein Alphabet und Z eine Menge von Zuständen. Dann heißt* $TS = (\Sigma, Z, z_0, \delta, \Omega)$ Transitionssystem*, wobei* $z_0 \in Z$ *der* Anfangszustand*,* $\delta \subseteq Z \times (\Sigma \cup \{\tau\}) \times Z$ *eine Menge von* Transitionen *und* $\Omega \subseteq Z$ *eine Menge von* Endzuständen *ist.*

Die Einschränkung eines Transitionssystems auf die vom Anfangszustand aus erreichbaren Zuständen nennen wir *Erreichbarkeitsgraph.*
Zwei Graphen heißen *isomorph* zueinander, wenn sie bijektiv aufeinander abgebildet werden können.

Definition 2.4 (Isomorphismus) *Seien* $TS_1 = (\Sigma, Z_1, z_{1,0}, \delta_1, \Omega_1)$ *und* $TS_2 = (\Sigma, Z_2, z_{2,0}, \delta_2, \Omega_2)$ *Transitionssysteme. Eine bijektive Funktion* $f : Z_1 \to Z_2$ *heißt* Isomorphismus *(zwischen* TS_1 *und* TS_2*), wenn für jedes Paar* $z, z' \in Z_1$ *gilt:* (z, a, z') *gdw.* $(f(z), a, f(z'))$*. Zwei Transitionssysteme heißen* isomorph*, wenn es einen Isomorphismus zwischen ihnen gibt.*



(a) A (b) Transitionssystem von A

Abbildung 2.2.: Ein Serviceautomat und sein Transitionssystem

Ein Serviceautomat kann kommunizierende und nicht-kommunizierende Aktionen ausführen. Das Ausführen der nicht-kommunizierenden Aktion τ entspricht einem internen Schritt. Das Ausführen einer kommunizierenden Aktion entspricht dem Kommunizieren einer Nachricht an einen anderen Service. Eine kommunizierende Aktion besteht aus einem Kanal und einer Belegung der Variablen dieses Kanals.

Die kommunizierende Aktionen bilden zusammen das *Alphabet* des Serviceautomaten.

Definition 2.5 (Aktion, Alphabet eines Serviceautomaten) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat. Sei $c \in C$ ein Kanal und β eine Belegung über $var(c)$. Dann heißt (c, β)* kommunizierende Aktion *von A.*

Die kommunizierenden Aktionen bilden das Alphabet $\Sigma(A)$ *des Serviceautomaten. Das spezielle Symbol τ heißt* nicht-kommunizierende Aktion *und ist nicht Teil des Alphabets. Eine Folge kommunizierender Aktionen $w \in \Sigma^*(A)$ nennen wir* Eingabewort *von A.*

Zur Vereinfachung der Notation nehmen wir an, dass die Variablen jedes Kanals geordnet sind. Damit können wir eine kommunizierende Aktion kompakt als Tupel schreiben, also etwa $a.(3, 5)$ für eine Aktion eines Kanals mit zwei Variablen, die mit den Zahlen 3 bzw. 5 belegt sind.

Ein Knoten und eine Belegung seiner Variablen bilden zusammen einen *Zustand* des Serviceautomaten.

Definition 2.6 (Zustand, Instanz) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Automat. Sei $q \in Q$, β eine Belegung von $var(q)$. Dann heißt $z = (q, \beta)$ ein* Zustand *von A. z nennen wir eine* Instanz *von q.*

Ein Zustand $z = (q, \beta)$ heißt Anfangszustand*, falls $q = q_0$ und* Endzustand*, falls $q \in \Omega$. Die Menge der Zustände von A ist $Z(A)$.*

Ein *Schaltmodus* einer Kante ist eine Belegung, die jeder Variable des Quellknotens und des Zielknotens der Kante sowie den Variablen des Kanals der Kante einen Wert zuordnet. Der Begriff des Schaltmodus ist den gefärbten Petrinetzen [38] entlehnt.

Definition 2.7 (Schaltmodus) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat und $e = (q, c, G, q')$ eine Kante von A. Eine Belegung $\alpha : var(e) \to \mathcal{U}$ nennen wir* Schaltmodus *von e.*

Ein Guard ist eine Teilmenge aller Schaltmodi einer Kante. Der Guard definiert, welche Instanzen des Quellknotens und des Zielknotens der Kante durch einen *Schritt* verbunden sind und welche Aktion in diesem Schritt ausgeführt wird.

Definition 2.8 (Schritt, Aktivierung) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat und $e = (q, c, G, q')$ eine Kante von A. Seien $z = (q, \beta)$, $z' = (q', \beta')$ Zustände von A und $\alpha \in G$ ein Schaltmodus von e mit $\alpha(x) = \beta(x)$ für $x \in var(q)$ und $\alpha(x^{\text{new}}) = \beta'(x)$ für $x^{\text{new}} \in \text{varnew}(var(q'))$.*

Dann heißt (z, e, α, z') Schritt *von A. Wir schreiben kurz $z \xrightarrow{e,\alpha}_A z'$ für einen Schritt.*

e heißt aktiviert *in z für einen Schaltmodus α von e, wenn es einen Zustand z' und einen Schritt $z \xrightarrow{e,\alpha}_A z'$ gibt.*

Jedem Schritt ist jeweils eindeutig eine Aktion zugeordnet.

Definition 2.9 (Aktion eines Schrittes) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat und $e = (q, c, G, q')$ eine Kante von A. Sei $z \xrightarrow{e,\alpha}_A z'$ ein Schritt von A.*

Sei $c \neq \tau$ ein Kanal, β die Einschränkung von α auf $var(c)$. Dann nennen wir $a = (c, \beta) \in \Sigma(A)$ die (kommunizierende) Aktion von e in Schaltmodus α.

Sei $c = \tau$. Dann nennen wir τ die (nicht-kommunizierende) Aktion von e in Schaltmodus α).

Wir schreiben kurz $z \xrightarrow{a}_A z'$, wenn es einen Schritt $z \xrightarrow{e,\alpha}_A z'$ gibt, so dass a die Aktion von e in α ist.

Das Transitionssystem eines Serviceautomaten ist der Graph, der aus den Zuständen und Schritten des Serviceautomaten gebildet wird. Jeder Schritt ist mit seiner jeweiligen Aktion beschriftet.

Definition 2.10 (Transitionssystem eines Serviceautomaten) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat. Das Transitionssystem von A ist das Transitionssystem $TS(A) = (\Sigma(A), Z(A), z_0, \delta, Z_\Omega)$, wobei*

- *z_0 der Anfangszustand von A,*
- *$\delta = \{(z, a, z') | z \xrightarrow{a}_A z'\}$ und*
- *Z_Ω die Menge der Instanzen der Endknoten von A.*

Abb. 2.2 zeigt das Transitionssystem eines Serviceautomaten A.

Ein Ablauf eines Serviceautomaten A ist eine Sequenz von Schritten. Die Sequenz der kommunizierenden Aktionen, die dieser Ablauf ausführt, nennen wir *Trace*.

Definition 2.11 (Ablauf, Trace) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat. Ein* Ablauf *r von A von einem Zustand z zu einem Zustand z' ist eine Folge $z \overset{e_1,\alpha_1}{\rightarrow}_A z_1 \overset{e_2,\alpha_2}{\rightarrow} \dots \overset{e_{n-1},\alpha_{n-1}}{\rightarrow}_A z_{n-1} \overset{e_n,\alpha_n}{\rightarrow}_A z'$ von Schritten von z nach z'. Dabei sind $z_1, \dots, z_{n-1} \in Z(A)$. Die Anzahl der Schritte n nennen wir die* Länge *des Ablaufes.*

Sei $a_i \in \Sigma(A) \cup \{\tau\}$ jeweils die Aktion des i-ten Schrittes.

Das Wort $w \in \Sigma^*(A)$, das aus $a_1, \dots, a_n$ durch Streichen der τ entsteht, nennen wir den Trace *des Ablaufes.*

Wir schreiben $z \overset{*}{\rightarrow}_A z'$, wenn es einen Ablauf von z zu z' gibt. Wir schreiben $z \overset{w}{\Longrightarrow}_A z'$, wenn es einen Ablauf von z zu z' mit dem Trace w gibt.

Einen Ablauf, der beim Anfangszustand von A beginnt, nennen wir einen Ablauf von A. Den Trace eines Ablaufes von A nennen wir einen Trace von A. Die Menge der Traces von A sei $\mathrm{Tr}_A(A)$.

Ein endlicher Automat kann als ein Spezialfall eines Serviceautomaten mit endlich vielen Zuständen und leerer Variablenmenge dargestellt werden. Ein *Zustandsautomat* ist das Äquivalent eines endlichen Automaten in unserem Formalismus. Ein Zustandsautomat kann im Gegensatz zu einem endlichen Automaten auch unendlich viele Zustände haben. Für jede Kante eines Zustandsautomaten gibt es genau einen Schaltmodus, in dem sie aktiviert ist.

Definition 2.12 (Zustandsautomat) *Ein Serviceautomat A heißt* Zustandsautomat, *wenn die Variablenmenge jedes seiner Knoten leer ist und $|G| = 1$ für jede Kante $e = (q, c, G, q')$ ist.*

Die Kanäle eines Zustandsautomaten dürfen im Gegensatz zu den Knoten Variablen haben.

Ein Zustandsautomat ist isomorph zu seinem Transitionssystem. Wir identifizieren daher einen Zustandsautomaten mit seinem Transitionssystem und unterscheiden nicht zwischen Knoten und Zuständen.

Zu jedem Serviceautomaten gibt es einen Zustandsautomaten mit isomorphem Transitionssystem. Aus dem Transitionssystem eines Serviceautomaten können wir einen isomorphen Zustandsautomaten gewinnen. Diesen nennen wir die *Entfaltung* des Serviceautomaten.

Definition 2.13 (Entfaltung eines Serviceautomaten) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat. Die* Entfaltung *von A ist der Zustandsautomat*
$\mathrm{unfold}(A) = (Z(A), C, var', z_0, E', Z_\Omega)$ *mit*

- *$var'(q) = \emptyset$ für $q \in Q$,*
- *$var'(c) = var(c)$ für jeden Kanal $c \in C$,*
- *z_0 der Anfangszustand von A,*
- *Z_Ω die Menge der Instanzen der Endknoten von A,*

- $E' = \{(z, c, G_a, z') | z \xrightarrow{a} z', a = (c, \beta) \in \Sigma(A) \cup \{\tau\}\}$, *wobei* $G_a = \{\beta_a\}$ *und* β_a *ist die Belegung der Variablen von* c *mit* $a = (c, \beta_a) \in \Sigma(A)$.

Einen Serviceautomaten, in dessen Transitionssystem alle ausgehenden Kanten eines Zustands verschiedene Beschriftungen haben und keine mit τ beschriftete Kante vorkommt, nennen wir *deterministisch*.

Definition 2.14 (Deterministischer Serviceautomat) *Ein Serviceautomat A heißt* deterministisch*, wenn keine Kante mit τ beschriftet ist und es für jeden erreichbaren Zustand z und jede kommunizierende Aktion $a \in \Sigma(A)$ höchstens einen Schritt $z \xrightarrow{e,\alpha} z'$ mit der Aktion a gibt.*

Ein weiterer wichtiger Spezialfall von Serviceautomaten sind *azyklische* Serviceautomaten.

Definition 2.15 (Azyklischer Serviceautomat, Tiefe) *Sei A ein Serviceautomat. Ein* Kreis *ist ein Pfad der Länge mindestens eins mit gleichem Anfangs- und Endknoten. A heißt* azyklisch*, wenn A keinen Kreis enthält. Die* Tiefe *von A ist die maximale Länge eines Pfades von A.*

Ein Spezialfall azyklischer Serviceautomaten sind baumförmige Serviceautomaten.

Definition 2.16 (Baumförmiger Serviceautomat) *Ein Serviceautomat A heißt* baumförmig*, wenn es vom Anfangsknoten zu jedem Knoten genau einen Pfad gibt.*

Zu einer bezüglich Präfixbildung abgeschlossenen Sprache L (d. h., zu jedem Wort aus L ist jeder Präfix des Wortes auch in L) über dem Alphabet eines Serviceautomaten können wir einen deterministischen, baumförmigen Zustandsautomaten mit der Tracemenge L bilden. Jedes Wort aus L korrespondiert zu genau einem Pfad des Baumes und umgekehrt.

Definition 2.17 (Kanonischer Baum) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat und $L \subseteq \Sigma^*(A)$ eine bezüglich Präfixbildung abgeschlossenen Sprache.*

Dann ist der kanonische Baum *(zu A) mit der Sprache L der Zustandsautomat $B = (L, C, var, \varepsilon, E, L)$ mit dem Transitionssystem so dass $w \xrightarrow{a}_B wa$ für jedes $w \in \Sigma^*$ und $a \in \Sigma$.*

Jeder Zustand des kanonischen Baumes ist nach Konvention ein Endzustand.

Wir benötigen ein Mittel, um das Verhalten von Serviceautomaten miteinander zu vergleichen. Dazu benutzen wir den von Milner [65] eingeführten Begriff der *schwachen Simulationsrelation* zwischen Serviceautomaten. Ein Serviceautomat B *simuliert* einen Serviceautomaten A, wenn er stets jede Aktion von A nachahmen kann.

Definition 2.18 (Schwache Simulationsrelation) *Seien A und B Serviceautomaten mit dem gleichen Alphabet Σ. Sei $\varrho \subseteq Z(A) \times Z(B)$ eine Relation, so dass für jedes Tupel*

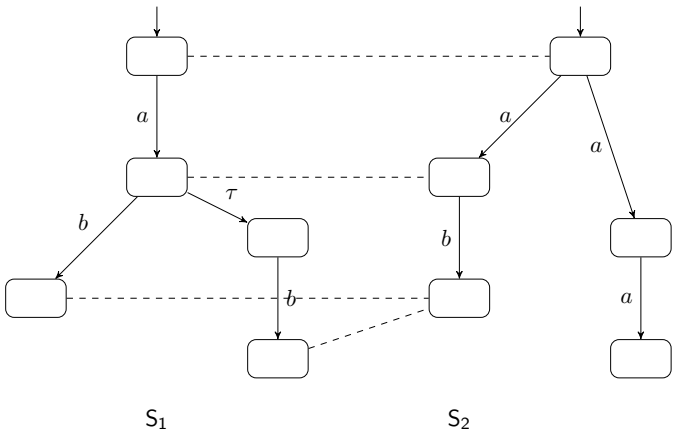


Abbildung 2.3.: Simulationsrelation. S_2 simuliert S_1, aber nicht umgekehrt

$(z_A, z_B) \in \varrho$ *gilt:*
Wenn $z_A \overset{w}{\Longrightarrow}_A z'_A$ *für ein* $w \in \Sigma^*$ *und* $z'_A \in Z_A$*, dann gibt es* $(z'_A, z'_B) \in \varrho$ *mit* $z_B \overset{w}{\Longrightarrow}_B z'_B$.

ϱ *heißt* schwache Simulationsrelation*, wenn die Anfangszustände* z_0^A, z_0^B *von* A *und* B *in Simulationsrelation stehen, d. h.* $(z_0^A, z_0^B) \in \varrho$*. Wir sagen* B simuliert A schwach*, wenn eine schwache Simulationsrelation zwischen* A *und* B *existiert.*

Wir bezeichnen für den Rest dieser Arbeit die schwache Simulation kurz als Simulation.

Die Simulationsbeziehung definiert eine verhaltensbasierte Ordnung auf der Menge der Serviceautomaten.

Lemma 2.1 (Quasiordnung) *Die Simulationsbeziehung zwischen Serviceautomaten definiert eine Quasiordnung auf der Menge der Serviceautomaten, d. h. sie ist reflexiv und transitiv.*

Statt „B simuliert A“ sagen wir alternativ auch B ist *permissiver* als A. Ein Serviceautomat ist ein permissivster Serviceautomat in einer Menge von Serviceautomaten, wenn er alle anderen Serviceautomaten der Menge simuliert.

2.3. Komposition von Serviceautomaten

Ein Service wird nicht in Isolation ausgeführt, sondern ist eingebunden in einen Zusammenschluss mehrerer Services, welche miteinander kommunizieren. Services kommunizieren über ihre *Nachrichtenkanäle* miteinander. Damit Services miteinander kommunizieren können, müssen ihre Nachrichtenkanäle miteinander verbunden werden. Das Verbinden mehrerer Services beschreiben wir formal durch die *Komposition* von Serviceautomaten.

Wir betrachten in dieser Arbeit zwei verschiedene Kommunikationsmodelle: *Synchrone* Kommunikation und *asynchrone* Kommunikation. Während das asynchrone Kommunikationsmodell sich gut eignet, um die Kommunikation von Services in einem verteilten System zu beschreiben, ist das synchrone Kommunikationsmodell formal wesentlich einfacher zu handhaben. Aus diesem Grund bildet das synchrone Kommunikationsmodell die Grundlage der in dieser Arbeit verwendeten formalen Definitionen.

Im asynchronen Kommunikationsmodell übernimmt stets ein Service, der *Sender* einer Nachricht, die Rolle des Initiators einer Kommunikation und ein anderer Service übernimmt die Rolle des *Empfängers* der Nachricht. Die Richtung des Informationsflusses ist im asynchronen Modell immer eindeutig bestimmt.

Im synchronen Kommunikationsmodell dagegen wird formal nicht unterschieden zwischen dem Sender und dem Empfänger einer Nachricht. Senden und Empfangen einer Nachricht finden gleichzeitig statt. Die am Nachrichtenaustausch beteiligten Services sind dabei gleichberechtigt. Es ist meist nicht möglich, dem Informationsfluß eine eindeutige Richtung zuzuordnen.

Häufig ist die Interaktion von Services miteinander intuitiv einfacher zu verstehen, wenn dem Nachrichtenaustausch eine Richtung und damit jedem Service eine bestimmte Rolle zugeordnet werden kann. Daher eigenen sich asynchron kommunizierende Services manchmal besser für Erläuterungen anhand von Beispielen.

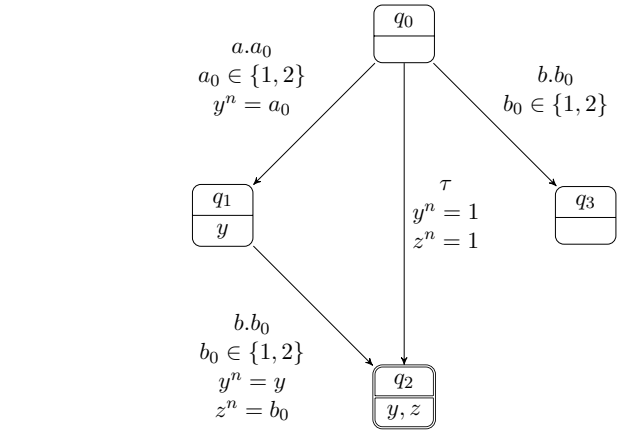
Die Komposition zweier asynchron kommunizierender Services kann formal auf die synchrone Komposition der beiden Services mit einem *Nachrichtenpuffer* zurückgeführt werden. Asynchrone Kommunikation wird in Abschnitt 2.3.1 eingeführt.

Die in dieser Arbeit behandelten Probleme sind im Kern unabhängig von der Art und Weise, wie Nachrichten kommuniziert werden. In einzelnen Fällen sind für asynchrone Kommunikationsmodell geringfügige Anpassungen gegenüber dem synchronen notwendig.

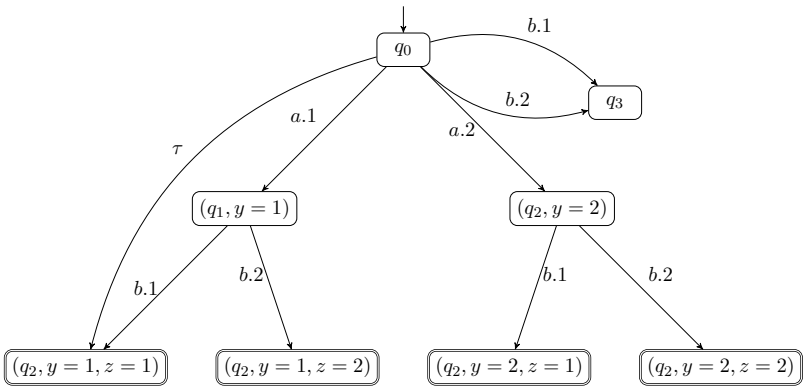
Wir stellen zunächst die *Komposition* synchron kommunizierender Serviceautomaten vor. Die synchrone Komposition zweier Serviceautomaten ist eine Variante des *Kreuzproduktes*, welches von den endlichen Automaten bekannt ist.

Das Kreuzprodukt zweier Zustandsautomaten entspricht genau dem Kreuzprodukt endlicher Automaten. Für Serviceautomaten mit Variablen definieren wir eine Verallgemeinerung des Kreuzproduktes. Das Transitionssystem des Kreuzproduktes zweier Serviceautomaten ist das Kreuzprodukt ihrer Entfaltungen.

Damit wir das Kreuzprodukt zweier Serviceautomaten bilden können, müssen diese *interfaceäquivalent* sein, d. h. sie müssen die gleichen Kanäle haben.



(a) B



(b) Transitionssystem von B

Abbildung 2.4.: Serviceautomat B

Definition 2.19 (Interfaceäquivalente Serviceautomaten) *Zwei Serviceautomaten A und B heißen* interfaceäquivalent*, wenn beide Serviceautomaten die gleichen Kanalmengen haben und jeder Kanal in beiden Serviceautomaten die gleiche Variablenmenge hat.*

Serviceautomaten sind genau dann interfaceäquivalent, wenn sie das gleiche Alphabet haben.

Aus formalen Gründen nehmen wir an, dass keine Variable eines Knotens eines der beiden Serviceautomaten gleichzeitig eine Variable eines Knotens des jeweils anderen Serviceautomaten ist. Damit kann jede Variable eines Knotens des Kreuzproduktes eindeutig einem der beiden Serviceautomaten zugeordnet werden. Wir nehmen in dieser Arbeit stets an, dass je zwei Serviceautomaten, deren Kreuzprodukt wir bilden, disjunkte Variablenmengen (mit Ausnahme der Variablen der Kanäle) haben, ohne dies explizit zu erwähnen. Dies ist keine Einschränkung, da Variablen immer geeignet umbenannt werden können.

Die Serviceautomaten A (Abb. 2.2) und B (Abb. 2.4) sind interfaceäquivalent. Abb. 2.5 zeigt das Kreuzprodukt von A und B.

Jeder Knoten des Kreuzproduktes $A \times B$ zweier Serviceautomaten A und B ist jeweils ein Paar von einem Knoten von A und einem Knoten von B. Die Variablenmenge des Knotens ist die Vereinigung der Variablen der beiden kombinierten Knoten.

Für jede τ-Kante von A oder B hat das Kreuzprodukt eine entsprechende τ Kante. Ein τ-Schritt von A ändert nicht den Knoten von B und die Belegung seiner Variablen. Die gilt analog für einen τ-Schritt von B.

Für ein Paar von Kanten von A und B, die mit dem gleichen Kanal c beschriftet sind, hat das Kreuzprodukt eine mit c beschriftete Kante, die sich aus der Kombination der beiden Kanten ergibt. Der Guard der neuen Kante ist die Konjunktion der Guards der beiden Kanten.

Die Kante des Kreuzproduktes ist daher genau dann ausführbar, wenn die Guards beider Kanten erfüllt sind. Da die Variablen des gemeinsamen Kanals c der kombinierten Kanten in beiden Guards vorkommen, kann die neue Kante nur dann ausgeführt werden, wenn diese Variablen in A und B gleich belegt sind, d. h. wenn A und B die gleiche Aktion ausführen können.

Definition 2.20 (Kreuzprodukt) *Seien $A = (Q_A, C, var_A, q_0^A, E_A, \Omega_A)$ und $B = (Q_B, C, var_B, q_0^B, E_B, \Omega_B)$ interfaceäquivalente Serviceautomaten. Keine Variable eines Knotens von A sei eine Variable eines Knotens von B und umgekehrt.*

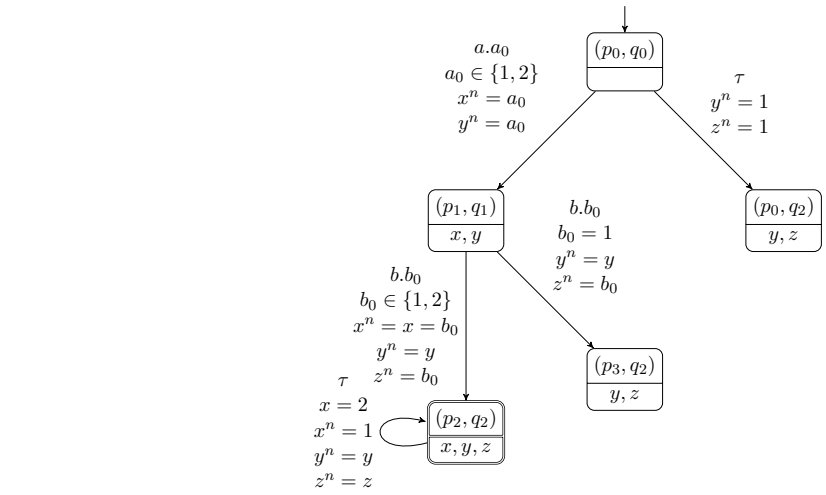
Das Kreuzprodukt *von A und B ist der Serviceautomat*

$$A \times B \underset{Def}{=} (Q_A \times Q_B, C, var, (q_0^A, q_0^B), E, \Omega_A \times \Omega_B)$$

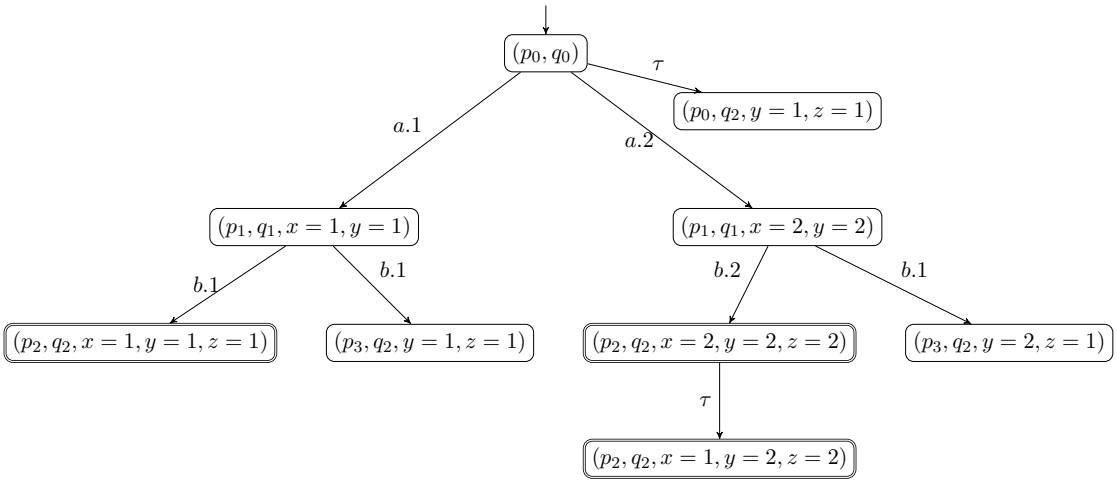
mit $var(q_A, q_B) = var_A(q_A) \cup var_B(q_B)$ für jeden Knoten $(q_A, q_B) \in Q_A \times Q_B$.

Die Kantenmenge ist die kleinste Menge E mit

- *τ-Schritt in A:*
  Wenn $(q_A, \tau, G, q'_A) \in E_A$,



(a) $\mathsf{A} \times \mathsf{B}$



(b) Transitionssystem von $\mathsf{A} \times \mathsf{B}$

Abbildung 2.5.: Kreuzprodukt $\mathsf{A} \times \mathsf{B}$ der Serviceautomaten aus Abb. 2.2 und Abb. 2.4 und dessen Transitionssystem

dann $((q_A, q_B), \tau, G', (q'_A, q_B)) \in E$,
wobei $G' = G \wedge \bigwedge_{x \in var(q_B)} x^{\text{new}} = x$

- *τ-Schritt in B:*
  Wenn $(q_B, \tau, G, q'_B) \in E_B$,
  dann $((q_A, q_B), \tau, G', (q_A, q'_B)) \in E$,
  wobei $G' = G \wedge \bigwedge_{x \in var(q_A)} x^{\text{new}} = x$

- *kommunizierender Schritt:*
  Wenn $c \in C$, $(q_A, c, G_A, q'_A) \in E_A$, $(q_B, c, G_B, q'_B) \in E_B$, *dann* $((q_A, q_B), c, G_A \wedge G_B, (q'_A, q'_B)) \in E$

Einen Zustand $z = ((q_A, q_B), \gamma)$ des Kreuzproduktes schreiben wir auch als $z = (q_A, q_B, \alpha, \beta)$, wobei α und β jeweils die Einschränkungen von γ auf die Variablen von q_A bzw. q_B sind. Wir schreiben z auch als das Paar $z = (z_A, z_B)$ der Zustände $z_A = (q_A, \alpha)$ und $z_B = (q_B, \beta)$ von A bzw. B.
Nach Definition des Kreuzproduktes folgt

Korollar 2.2 *Für das Kreuzprodukt zweier interfaceäquivalenter Serviceautomaten A und B gilt*

- $(q_A, q_B, \alpha, \beta) \xrightarrow{\tau}_{A \times B} (q'_A, q'_B, \alpha', \beta')$ *gdw.*
  - $(q_A, \alpha) \xrightarrow{\tau}_A (q'_A, \alpha')$ *und* $(q'_B, \beta') = (q_B, \beta)$ *oder*
  - $(q'_A, \alpha') = (q_A, \alpha)$ *und* $(q_B, \beta) \xrightarrow{\tau}_B (q'_B, \beta')$

- $(q_A, q_B, \alpha, \beta) \xrightarrow{a}_{A \times B} (q'_A, q'_B, \alpha', \beta')$ *gdw.*
  $(q_A, \alpha) \xrightarrow{a}_A (q'_A, \alpha')$ *und* $(q_B, \beta) \xrightarrow{a}_B (q'_B, \beta')$
  für $a \in \Sigma(A) = \Sigma(B)$.

Die Entfaltung des Kreuzproduktes zweier Serviceautomaten ist das Kreuzprodukt der Entfaltungen der beiden Serviceautomaten nach der üblichen Definition des Kreuzproduktes für endliche Automaten.

Lemma 2.3 (Kreuzprodukt der Entfaltung) *Seien A, B interfaceäquivalente Serviceautomaten. Dann ist* $\text{unfold}(A \times B)$ *isomorph zu* $\text{unfold}(A) \times \text{unfold}(B)$.

Abläufe zweier Serviceautomaten sind genau dann gemeinsam im Kreuzprodukt ausführbar, wenn sie den gleichen Trace haben.

Lemma 2.4 *Seien A, B interfaceäquivalente Serviceautomaten. Dann gilt* $(z_A, z_B) \xrightarrow{*}_{A \times B} (z'_A, z'_B)$ *gdw.* $z_A \overset{w}{\Longrightarrow}_A z'_A$ *und* $z_B \overset{w}{\Longrightarrow}_B z'_B$.

Das Kreuzprodukt zweier Serviceautomaten verallgemeinern wir zur *synchronen Komposition* zweier Serviceautomaten. Bei der synchronen Komposition müssen die Kanalmengen der komponierten Serviceautomaten im Unterschied zum Kreuzprodukt nicht

gleich sein. Nachrichtenkanäle, die nur in einem der komponierten Serviceautomaten vorkommen, werden nicht miteinander synchronisiert. Mit der Synchronen Komposition können im Unterschied zum Kreuzprodukt Kompositionen von mehr als zwei Services beschrieben werden.

Definition 2.21 (Interfacekompatible Serviceautomaten) *Zwei Serviceautomaten heißen* interfacekompatibel*, wenn jeder Kanal, der in beiden Serviceautomaten vorhanden ist, in beiden Serviceautomaten die gleiche Variablenmenge hat.*

Die synchrone Komposition zweier interfacekompatibler Serviceautomaten ist wieder ein Serviceautomat. Kanäle, welche die komponierten Serviceautomaten teilen, werden miteinander verschmolzen und aus dem Interface entfernt. Schritte der Serviceautomaten, welche über diese Kanäle kommunizieren, werden zu internen, nicht kommunizierenden Schritten der Komposition. Der Guard einer Kante, die über einen von den Serviceautomaten geteilten Kanal kommuniziert, wird eingeschränkt auf die Menge seiner Variablen ohne die Variablen des entfernten Kanals. Diejenigen Kanäle, welche von den komponierten Serviceautomaten nicht geteilt werden, bilden das Interface des durch die Komposition neu entstandenen Serviceautomaten. Schritte, die über nicht geteilte Kanäle kommunizieren, werden unabhängig voneinander ausgeführt.

Definition 2.22 (Synchrone Komposition) *Seien* $A = (Q_A, C_A, var_A, q_0^A, E_A, \Omega_A)$ *und* $B = (Q_B, C_B, var_B, q_0^B, E_B, \Omega_B)$ *interfacekompatible Serviceautomaten. Keine Variable eines Knotens von A sei eine Variable eines Knotens von B und umgekehrt.*

Die synchrone Komposition *von A und B ist der Serviceautomat*

$$A \otimes B \underset{Def}{=} (Q_A \times Q_B, C_A \triangle C_B, var, (q_0^A, q_0^B), E, \Omega_A \times \Omega_B)$$

mit $var(q_A, q_B) \underset{Def}{=} var_A(q_A) \cup var_B(q_B)$ *für jeden Knoten* $(q_A, q_B) \in Q_A \times Q_B$.

Die Kantenmenge ist die kleinste Menge E mit

- *Schritt in A:*
  wenn $c \in (C_A \setminus C_B) \cup \{\tau\}$ *und* $(q_A, c, G, q'_A) \in E_A$,
  dann $((q_A, q_B), c, G', (q'_A, q_B)) \in E$
  wobei $G' = G \wedge \bigwedge_{x \in var(q_B)} x^{\text{new}} = x$

- *Schritt in B:*
  wenn $c \in (C_B \setminus C_A) \cup \{\tau\}$ *und* $(q_B, B, G, q'_B) \in E_B$,
  dann $((q_A, q_B), c, G', (q_A, q'_B)) \in E$
  wobei $G' = G \wedge \bigwedge_{x \in var(q_A)} x^{\text{new}} = x$

- *kommunizierender Schritt:*
  wenn $c \in C_A \cap C_B$, $(q_A, c, G_A, q'_A) \in E_A$ *und* $(q_B, c, G_B, q'_B) \in E_B$,
  dann $((q_A, q_B), \tau, G', (q'_A, q'_B)) \in E$
  wobei $G' = (G_A \wedge G_B)_{|X}$ *mit* $X = var(q_A, q_B) \cup \text{varnew}(var(q_A, q_B))$.

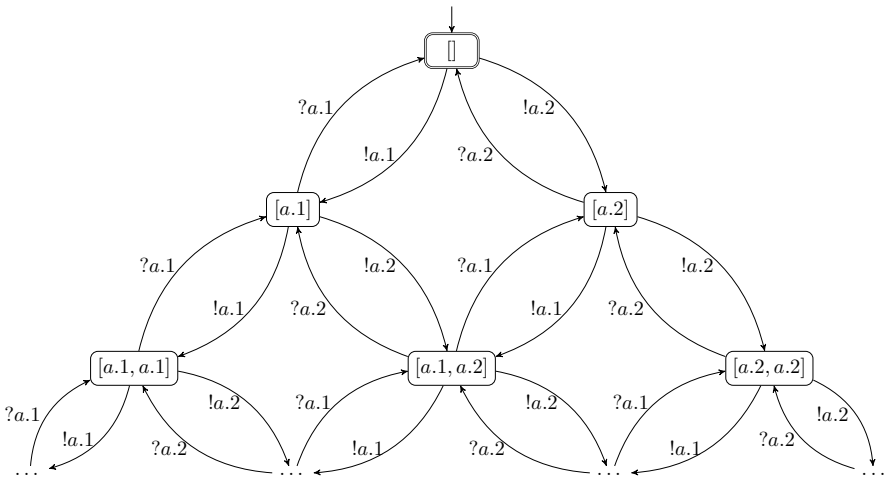


Abbildung 2.6.: Ein ungeordneter Nachrichtenpuffer ohne Kapazitätsbeschränkung

In dieser Arbeit betrachten wir überwiegend die Komposition von genau zwei Serviceautomaten. Die formale Unterscheidung zwischen Kreuzprodukt und synchroner Komposition ist in diesem Szenario unerheblich. In den formalen Definitionen werden wir daher stets mit dem Kreuzprodukt statt mit der Komposition arbeiten.

Formal benötigen wir die Komposition zum Komponieren von Nachrichtenpuffern und Adaptern.

2.3.1. Asynchron kommunizierende Serviceautomaten

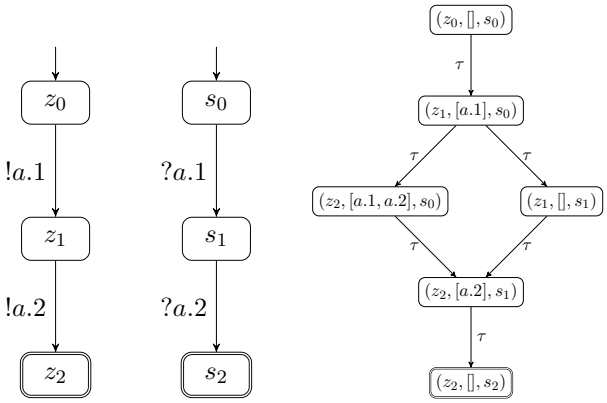
Im in Abschnitt 2.3 vorgestellten synchronen Kommunikationsmodell findet das Senden und Empfangen einer Nachricht gleichzeitig statt. Ein System, in welchem sich die miteinander kommunizierenden Prozesse an ein- und demselben Ort befinden, kann mit diesem Modell gut beschrieben werden.

In einem verteilten System befinden sich Sender und Empfänger in der Regel an verschiedenen Orten, so dass zwischen dem Senden und dem Empfangen einer Nachricht immer eine gewisse Zeit vergeht, in welcher die Nachricht unterwegs ist. Ein asynchrones Kommunikationsmodell, in dem zwischen *Senden* und *Empfangen* einer Nachricht unterschieden wird, eignet sich daher besser, um ein verteiltes System zu beschreiben.

Das Senden und Empfangen einer Nachricht beschreiben wir formal durch das Schreiben und Lesen in und aus einem *Nachrichtenpuffer*. Diesen Nachrichtenpuffer komponieren wir synchron mit den Serviceautomaten, welche er verbindet.

In diesem Abschnitt illustrieren wir, wie asynchrone Kommunikation auf synchrone Kommunikation zurückgeführt werden kann. Diese Reduktion stellen wir vor, da die in dieser Arbeit behandelten Begriffe und Probleme ursprünglich für Modelle asynchron kommunizierender Services eingeführt wurden. Formal spielt asynchrone Kommunikation im Rahmen dieser Arbeit keine zentrale Rolle.

Abb. 2.6 zeigt einen Nachrichtenpuffer für einen Kanal a und ein zweielementiges



(a) C (b) D (c) Komposition von C und D

Abbildung 2.7.: Asynchron kommunizierende Serviceautomaten und ihre Komposition

Universum $\mathcal{U} = \{1, 2\}$. Ein Nachrichtenpuffer hat je einen Kanal zum Senden (notiert mit vorangestelltem Ausrufezeichen !) und zum Empfangen (notiert mit vorangestelltem Fragezeichen ?) von Nachrichten. Sende- und Empfangskanal haben die gleiche Variablenmenge. Jeder Zustand eines Nachrichtenpuffers ist eine Multimenge von Nachrichten. Eine Nachricht ist eine Belegung der Variablen der Kanäle. Eine Nachricht notieren wir in Tupelschreibweise analog zu der Notation, die wir für Aktionen verwenden. Dem Tupel stellen wir den Namen des Nachrichtenpuffers voran. Die Aktionen des Sendekanals fügen jeweils eine Nachricht zur Multimenge hinzu, die des Empfangskanals entfernen eine Nachricht. Der Nachrichtenpuffer ist genau dann in seinem Endzustand, wenn er leer ist.

Abb. 2.7 zeigt zwei asynchron kommunizierende Serviceautomaten. Die Kanäle der Serviceautomaten sind jeweils zueinander komplementär, so dass jeweils ein Serviceautomat Nachrichten in dem Puffer schreibt und der andere Nachrichten aus dem Puffer entfernt. Der linke Serviceautomat C hat einen Sendekanal $!a$, der rechte Serviceautomat D hat den dazu komplementären Empfangskanal $?a$. Die asynchrone Komposition der Serviceautomaten wird gebildet, indem beide synchron mit dem Nachrichtenpuffer komponiert werden.

Ein *asynchron kommunizierender Serviceautomat* ist ein Serviceautomat zusammen mit einer Menge von Nachrichtenpuffern, so dass jeder Kanal des Serviceautomaten jeweils entweder ein Sendekanal oder ein Empfangskanal eines der Nachrichtenpuffer ist. Zwei asynchron kommunizierende Serviceautomaten sind kompatibel, wenn sie die gleiche Menge von Nachrichtenpuffern haben und jeweils komplementäre Kanäle eines Nachrichtenpuffers benutzen. Kompatible asynchron kommunizierende Serviceautomaten werden komponiert, indem sie synchron miteinander und allen ihren Nachrichtenpuffern komponiert werden.

Wir nehmen in diesem asynchronen Kommunikationsmodell an, dass ein Nachrichtenpuffer unbegrenzt groß ist. Mit einem unbegrenzt großen Nachrichtenpuffer ist es immer

möglich, Nachrichten zu senden, ohne einen Pufferüberlauf zu verursachen oder ältere Nachrichten zu überschreiben. Weiterhin nehmen wir an, dass Nachrichten ungeordnet im Nachrichtenpuffer gespeichert werden. Nachrichten können sich also gegenseitig auf dem Weg zum Empfänger überholen.

Dieses asynchrone Kommunikationsmodell entspricht dem Kommunikationsmodell der endlichen Automaten, die in [57] zur Beschreibung von Services verwendet werden. Einige der in dieser Arbeit verwendeten Begriffe und Techniken Konzepte wurden ursprünglich für dieses Servicemodell definiert. Offene Netze [31, 6], kommunizieren ebenfalls nach diesem Modell. Ein offenes Netz mit endlichem Erreichbarkeitsgraphen kann in einen äquivalenten asynchron kommunizierenden Serviceautomaten umgewandelt werden.

Daneben gibt es weitere asynchrone Kommunikationsmodelle, die geordnete Nachrichtenpuffer verwenden oder sich in der Pufferkapazität unterscheiden. Lohmann [55] stellt eine Klassifizierung unterschiedlicher Kommunikationsmodelle vor.

2.3.2. Wissensfunktion

In diesem Abschnitt führen wir ein grundlegendes Konzept zur Analyse eines verteilten Systems ein, welches das *Wissen* eines Service über einen anderen Service beschreibt.

Die Services eines verteilten Systems haben im allgemeinen nur unvollständiges Wissen über den Zustand des Gesamtsystems. Jeder Service kennt seinen eigenen Zustand, nicht aber den Zustand der Services, mit denen er kommuniziert.

Jeder Service verhält sich aus Sicht eines anderen Service wie eine „black box“. Die Zustandsübergänge, die ein Service intern ausführt, sind für den jeweils anderen Service nicht sichtbar. Für die Umgebung eines Service sind lediglich die von diesem gesendeten Nachrichten sichtbar. Bei einem nichtdeterministischen Serviceautomaten kann einer Nachricht nicht eindeutig ein Zustandsübergang zugeordnet werden. Daher kann anhand der von einem Service gesendeten Nachrichtensequenz nicht eindeutig bestimmt werden, in welchem Zustand dieser sich gerade befindet.

Durch Beobachtung der kommunizierten Nachrichten kann jedoch die Menge der Zustände, in der sich ein Serviceautomat möglicherweise befindet, eingeschränkt werden.

Jedem Eingabewort w eines Serviceautomaten ordnen wir die Menge der Zustände zu, die über einen Ablauf erreichbar sind, der dieses Wort als Trace hat. Diese Menge ist das *Wissen* der Umgebung des Serviceautomaten nach Eingabe w.

Definition 2.23 (Wissen eines Eingabewortes) *Sei A ein Serviceautomat mit Anfangszustand z_0 und $w \in \Sigma^*(A)$ ein Eingabewort von A. Dann ist*

$$\mathrm{know}_A(w) \underset{Def}{=} \{z | z_0 \stackrel{w}{\Longrightarrow}_A z\}$$

das Wissen *über A für die Eingabe w.*

Jedem Trace des Serviceautomaten ist mindestens ein Zustand zugeordnet.

Korollar 2.5 *Für jedes Eingabewort $w \in \Sigma(A)^*$ eines Serviceautomaten A gilt*

$$\text{know}_A(w) \neq \emptyset \text{ gdw. } w \in \text{Tr}(A)$$

Für einige ausgewählte Traces des Serviceautomaten A aus Abb. 2.2 erhalten wir die folgenden Mengen.

$$\begin{aligned} \text{know}_A(\varepsilon) &= \{p_0\} \\ \text{know}_A(a.1\, b.1) &= \{(p_2, x = 1), p_3\} \\ \text{know}_A(a.2\, b.2) &= \{(p_2, x = 1), (p_2, x = 2)\} \end{aligned}$$

In dieser Arbeit untersuchen wir vornehmlich die Interaktion zwischen jeweils genau zwei miteinander komponierten Services. In einem verteilten System mit genau zwei Services können wir jedem Service zuordnen, was dieser Service in einem seiner Zustände über den jeweils anderen Service weiß.

Das *Wissen* eines Zustandes z eines Serviceautomaten B über den Serviceautomaten A ist die Menge der Zustände des Serviceautomaten A, in denen A sich befinden kann, wenn B sich in Zustand z befindet.

Definition 2.24 (Wissen eines Zustandes) *Seien A, B interfaceäquivalente Serviceautomaten. Das* Wissen *von B über A in einem Zustand z_B von B ist die Menge*

$$\text{know}_{A,B}(z_B) \underset{Def}{=} \{z_A \in Z(A), (z_A, z_B) \text{ erreichbar in } A \times B\}$$

Aus dem Transitionssystem von $\mathsf{A} \times \mathsf{B}$ erhalten wir für ausgewählte Zustände von B die folgenden Wissensmengen.

$$\begin{aligned} \text{know}_{\mathsf{A},\mathsf{B}}(q_0) &= \{p_0\} \\ \text{know}_{\mathsf{A},\mathsf{B}}(q_2, y = 1, z = 1) &= \{p_0, (p_2, x = 1), p_3\} \\ \text{know}_{\mathsf{A},\mathsf{B}}(q_2, y = 2, z = 2) &= \{(p_2, x = 2), (p_2, x = 1)\} \end{aligned}$$

Das Wissen eines Zustands eines Serviceautomaten B über einen Serviceautomaten A ist die Vereinigung des Wissens der Traces der Abläufe, über die der Zustand erreichbar ist.

Korollar 2.6 *Seien A, B interfaceäquivalente Serviceautomaten und z_B ein Zustand von B. Dann gilt*

$$\text{know}_{A,B}(z_B) = \bigcup_{w \in \text{Tr}(z_B)} \text{know}_A(w)$$

Beweis Folgt aus Def. und Lemma 2.4. □

Es ist z. B. $\mathrm{Tr}(q_2, y = 1, z = 1) = \{\varepsilon, a.1\ b.1\}$. Das Wissen des Zustandes $(q_2, y = 1, z = 1)$ über A ergibt sich aus dem Wissen seiner beiden Traces.

$$\begin{aligned}\mathrm{know}_{A,B}(q_2, y = 1, z = 1) &= \mathrm{know}_A(\varepsilon) \cup \mathrm{know}_A(a.1\ b.1) \\ &= \{p_0\} \cup \{(p_2, x = 1), p_3\}\end{aligned}$$

Das Wissen jedes Eingabewortes kann schrittweise mittels Induktion berechnet werden. Dazu berechnet man zunächst $\mathrm{know}_A(\varepsilon)$ und wendet auf diese Zustandsmenge wiederholt die Funktion event an. Die Funktion event ordnet einer Zustandsmenge die Menge der Zustände zu, die von einem beliebigen Zustand der Menge über eine kommunizierende Aktion a und beliebig viele nicht-kommunizierende Schritte erreichbar sind.

Definition 2.25 (Event) *Sei A ein Serviceautomat, $Z' \subseteq Z(A), a \in \Sigma$. Dann sei*

$$\mathrm{event}_A(Z', a) \underset{Def}{=} \{z' | z \in Z, z \stackrel{a}{\Longrightarrow}_A z'\}$$

Die Funktion event_A entspricht der Kombination der closure und event genannten Funktionen aus [57].

Korollar 2.7 *Sei A ein Serviceautomat, $w \in \Sigma^*(A)$ und $a \in \Sigma(A)$. Dann gilt*

$$\mathrm{event}_A(\mathrm{know}_A(w), a) = \mathrm{know}_A(w\,a)$$

Aus Korollar 2.7 folgt insbesondere, dass das Anhängen einer Aktion a an Eingabeworte mit gleichem Wissen wieder zu Eingabeworten mit gleichem Wissen führt.

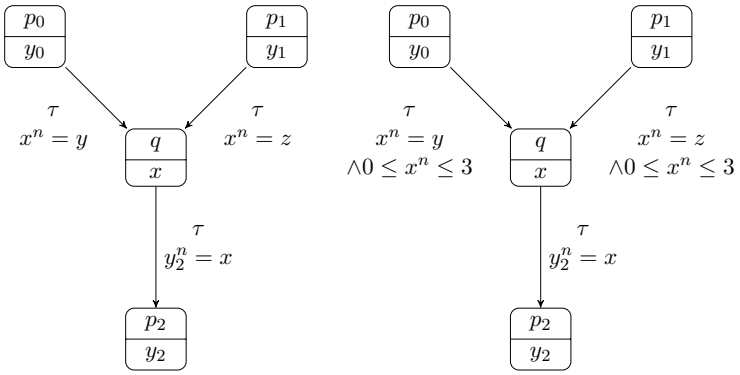
Korollar 2.8 *Sei A ein Serviceautomat mit Alphabet Σ. Dann gilt für $w, w' \in \Sigma^*$ und $a \in \Sigma$ mit $wa, w'a \in \mathrm{Tr}(A)$: Wenn $\mathrm{know}_A(w) = \mathrm{know}_A(w')$, dann $\mathrm{know}_A(wa) = \mathrm{know}_A(w'a)$.*

2.4. Weitere Operationen auf Serviceautomaten

In diesem Abschnitt stellen wir einige einfache Operationen zum Verändern von Serviceautomaten vor. Der Leser kann diesen Abschnitt zunächst überspringen und die Operationen bei Bedarf dann nachschlagen, wenn sie in späteren Kapiteln benötigt werden.

2.4.1. Entfernen von Knoten und Zuständen

Unter der *Einschränkung* eines Zustandsautomaten auf eine Zustandsmenge Z' verstehen wir den Zustandsautomaten, der aus diesem durch Entfernen aller Zustände entsteht, die nicht in Z' sind. Für Serviceautomaten, die keine Zustandsautomaten sind, definieren wir zwei verschiedene Verallgemeinerungen der Einschränkung. Beide Verallgemeinerungen



(a) q vor Einschränkung (b) q nach Einschränkung

Abbildung 2.8.: Einschränkung eines Knotens q auf die Zustandsmenge $M = \{(q, x = i) | 0 \leq x \leq 3\}$

entsprechen semantisch einer Einschränkung der Entfaltung des Serviceautomaten, auf den sie angewendet werden.

Die Einschränkung eines Serviceautomaten auf eine Knotenmenge Q' ist der Serviceautomat, der aus diesem durch Entfernen aller Zustände entsteht, die nicht in Q' sind. Aus technischen Gründen verlangen wir, dass der Anfangsknoten stets in Q' enthalten ist, da sein Serviceautomat stets einen Anfangsknoten haben muss.

Definition 2.26 (Einschränkung auf Knotenmenge) *Sei A ein Serviceautomat mit einer Knotenmenge Q. Q' sei eine Teilmenge von Q, die mindestens den Anfangszustand enthält. Die* Einschränkung *von A auf eine Knotenmenge Q' bezeichne den Serviceautomaten, der aus A durch Entfernen aller Knoten entsteht, die nicht in Q' sind. Das Entfernen eines Knotens schließe dabei das Entfernen angrenzender Kanten ein.*

Die Einschränkung eines Serviceautomaten auf eine Zustandsmenge Z' bezeichnet dagegen einen Serviceautomat, dessen Guards so angepasst werden, dass nur Zustände aus der Schnittmenge der erreichbaren Zustände mit Z' erreichbar sind.

Definition 2.27 (Einschränkung auf Zustandsmenge) *Seien A ein Serviceautomat, q ein Knoten von A, der nicht der Anfangsknoten von A ist und M eine Menge von Instanzen von q. Sei A' der Serviceautomat A', der aus A entsteht, indem jeder Guard G einer eingehenden Kante von q durch den Guard*

$$G' = \{\gamma \in G | es\ ex.\ (q, \beta) \in M\ mit\ \gamma(x^{new}) = \beta(x)\ für\ x \in var(x)\}$$

ersetzt wird. Dann sagen wir A' entsteht aus A durch Einschränkung von q auf M.

In syntaktischer Darstellung entspricht das Einschränken eines Knoten dem Ersetzen der Guards seiner eingehenden Kanten mit ihrer Konjunktion mit einer Formel, welche die Menge der Belegungen der Zustände repräsentiert, auf die der Knoten eingeschränkt

wird. Die Einschränkung eines Knotens q eines Serviceautomaten auf eine Menge M von Instanzen von q entspricht in der Entfaltung des Serviceautomaten dem Entfernen aller Instanzen von q, die nicht in M enthalten sind.

Abb. 2.8 zeigt die Einschränkung eines Knotens q auf die Menge seiner Instanzen, in denen x mit ganzen Zahlen zwischen 0 und 3 belegt ist.

2.4.2. Verschmelzen von Knoten

Unter dem Verschmelzen von Knoten eines Serviceautomaten verstehen wir das Zusammenfassen von Knoten zu einem neuen Knoten. Diese Operation ist aus der Graphentheorie auch als das *Kontrahieren* von Knoten bekannt. Beim Verschmelzen werden zwei oder mehrere Knoten durch einen neuen Knoten ersetzt. Die zu den verschmolzenen Knoten adjazenten Kanten werden dabei ebenfalls zusammengefasst, so dass diese zum neu entstandenen Knoten adjazent sind.

Wir definieren die Verschmelzungsoperation von Knoten zunächst für einzelne Zustände eines Zustandsautomaten. Anschließend verallgemeinern wir die Operation auf Knoten von Serviceautomaten.

Definition 2.28 (Verschmelzen von Zuständen) *Sei A ein Zustandsautomat mit Zustandsmenge Z und $z_1, z_2 \in Z$. Der Zustandsautomat A', der aus A durch Verschmelzen von z_1 und z_2 zu einem neuen Zustand z_{12} entsteht, sei wie folgt definiert:*

- *A' habe die Zustandsmenge $Z' = (Z \cup \{z_{12}\}) \setminus \{z_1, z_2\}$.*
- *A' habe die gleichen Kanäle wie A.*
- *z_{12} sei der Anfangszustand von A' gdw. z_1 oder z_2 der Anfangszustand von A ist. Andernfalls habe A' den gleichen Anfangszustand wie A.*
- *z_{12} sei ein Endzustand von A' gdw. z_1 oder z_2 ein Endzustand von A ist. Auf allen anderen Zuständen stimme die Endzustandsmenge von A' mit der von A überein.*

Die Kantenmenge von A' sei folgendermaßen definiert: Für $z, z' \in Z \setminus \{z_1, z_2\}$ sei

- $z \xrightarrow{a}_{A'} z'$ *gdw.* $z \xrightarrow{a}_{A} z'$
- $z \xrightarrow{a}_{A'} z_{12}$ *gdw.* $z \xrightarrow{a}_{A} z_1$ *oder* $z \xrightarrow{a}_{A} z_2$
- $z_{12} \xrightarrow{a}_{A'} z$ *gdw.* $z_1 \xrightarrow{a}_{A} z$ *oder* $z_2 \xrightarrow{a}_{A} z$
- $z_{12} \xrightarrow{a}_{A'} z_{12}$ *gdw. es ex.* $i, j \in \{1, 2\}$ *mit* $z_i \xrightarrow{a}_{A} z_j$

Da beim Verschmelzen mehrerer Zustände die Reihenfolge, in der diese verschmolzen werden, keine Rolle spielt, können wir das Verschmelzen zweier Zustände kanonisch auf das Verschmelzen von Mengen von Zuständen verallgemeinern.

Knoten eines Serviceautomaten werden in der gleichen Weise verschmolzen wie einzelne Zustände. Allerdings können jeweils nur Knoten mit gleicher Variablenmenge verschmolzen werden. Die Variablenmenge des entstehenden Knotens ist gleich der Menge

der Variablen der Knoten, aus denen er entsteht. Dadurch ist gewährleistet, dass die Variablenmenge zu den in den Guards der adjazenten Kanten benutzten Variablen passt.

Für Knoten mit unterschiedlicher Variablenmenge ist die Verschmelzungsoperation nicht definiert. Die Variablen von Knoten, welche die gleiche Anzahl von Variablen haben, können gegebenenfalls so umbenannt werden, das ein Verschmelzen möglich ist. Für jede Möglichkeit, Variablen durch Umbenennung aufeinander abzubilden, entsteht nach dem Verschmelzen im allgemeinen ein anderes Transitionssystem.

Definition 2.29 (Verschmelzen von Knoten) *Sei $A = (Q, C, var, q_0, E, \Omega)$ ein Serviceautomat und q_1, q_2 Knoten von A mit $var(q_1) = var(q_2)$. Der Serviceautomat $A' = (Q', C, var', q_0', E', \Omega')$, der aus A durch Verschmelzen von q_1 und q_2 zu einem Knoten q_{12} entsteht, sei wie folgt definiert:*

- $Q' = (Q \cup \{q_{12}\}) \setminus \{q_1, q_2\}$.
- $var'(q) = var(q)$ *für* $q \in Q \setminus \{q_1, q_2\}$ *und* $var'(q_{12}) = var(q_1) = var(q_2)$,
- $q_0' = q_{12}$ *falls* $q_0 = q_1$ *oder* $q_0 = q_2$, *andernfalls sei* $q_0' = q_0$.
- $\Omega' = \Omega \cup \{q_{12}\}$, *falls* $q_1 \in \Omega$ *oder* $q_2 \in \Omega$, $\Omega' = \Omega$, *sonst.*

Die Kantenmenge E' sei wie folgt definiert: Für $q, q' \in Q \setminus \{q_1, q_2\}$ gelte

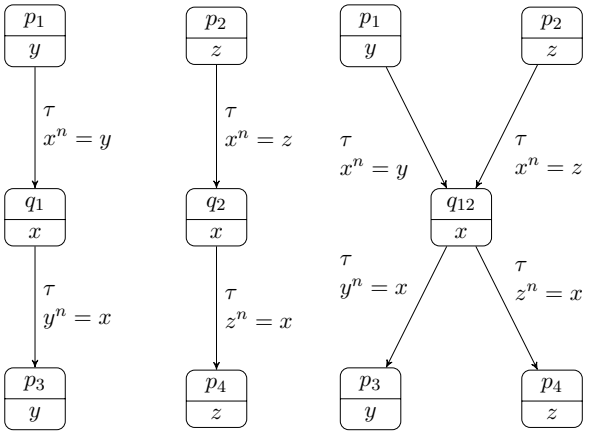
- $(q, c, G, q') \in E'$ *gdw.* $(q, c, G, q') \in E$
- $(q, c, G, q_{12}) \in E'$ *gdw. es ex.* $i \in \{1, 2\}$ *mit* $(q, c, G, q_i) \in E$
- $(q_{12}, c, G, q) \in E'$ *gdw. es ex.* $i \in \{1, 2\}$ *mit* $(q_i, c, G, q) \in E$
- $(q_{12}, c, G, q_{12}) \in E'$ *gdw. es ex.* $i, j \in \{1, 2\}$ *mit* $(q_i, c, G, q_j) \in E$

Abb. 2.9 zeigt einen Teil eines Serviceautomaten (ohne Anfangsknoten) vor und nach dem Verschmelzen zweier Knoten.

Das Verschmelzen zweier Knoten eines Serviceautomaten entspricht im Transitionssystem des Serviceautomaten dem paarweisen Verschmelzen von Instanzen der beiden Knoten. Dabei werden jeweils Zustände mit gleicher Variablenbelegung verschmolzen.

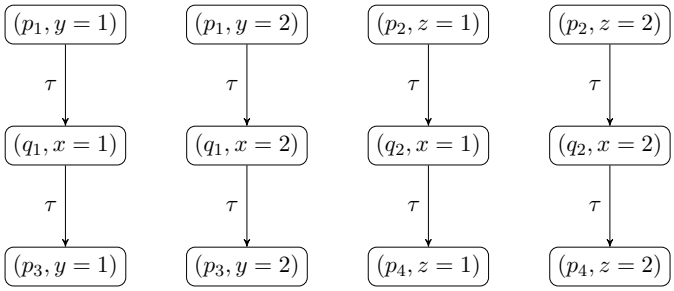
Korollar 2.9 *Seien A, A', q_1, q_2 und q_{12} wie in Def. 2.29. Dann ist die Entfaltung von A' isomorph zu dem Zustandsautomaten, der aus der Entfaltung von A entsteht, indem für jede Belegung β die Zustände (q_1, β) und (q_2, β) zu jeweils einem neuen Zustand verschmolzen werden. Der neue Zustand entspricht jeweils dem Zustand (q_{12}, β) von A'.*

Beispiel Das Verschmelzen der Knoten q_1 und q_2 des Serviceautomaten aus Abb. 2.9 resultiert im Transitionssystem (Abb. 2.10) im Verschmelzen der beiden Zustände $(q_1, x = 1)$ und $(q_2, x = 1)$ zu $(q_{12}, x = 1)$. Analog werden die beiden Zustände $(q_1, x = 2)$ und $(q_2, x = 2)$ zu $(q_{12}, x = 2)$ verschmolzen.

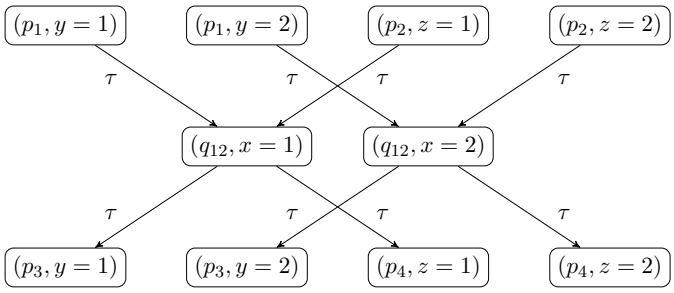


(a) vor Verschmelzen von q_1 und q_2 (b) nach Verschmelzen von q_1 und q_2

Abbildung 2.9.: Verschmelzen von Knoten

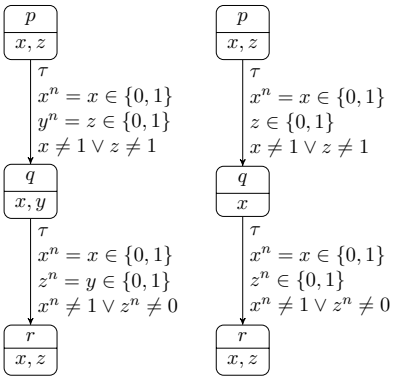


(a) vor Verschmelzen von q_1 und q_2



(b) nach Verschmelzen von q_1 und q_2

Abbildung 2.10.: Transitionssystem (Ausschnitt) des Serviceautomaten in Abb. 2.9



(a) vor Elimination (b) nach Elimination

Abbildung 2.11.: Elimination der Variablen y in q

2.4.3. Elimination einer Variablen

Das *Eliminieren* einer Variablen aus einem Knoten bezeichnet eine Operation, welche eine Variable aus der Variablenmenge eines Knotens entfernt. Die Guards der adjazenten Kanten des Knotens werden beim Eliminieren auf die neue Variablenmenge des Knotens eingeschränkt. Die eingeschränkten Guards hängen nicht mehr von der eliminierten Variablen ab.

Abb. 2.11 zeigt einen Ausschnitt eines Serviceautomaten und dessen Knoten q vor und nach Elimination seiner Variablen y.

Definition 2.30 (Eliminination einer Variablen) *Sei x die Variable eines Knotens q eines Serviceautomaten $A = (Q, C, var, q_0, E, \Omega)$.*

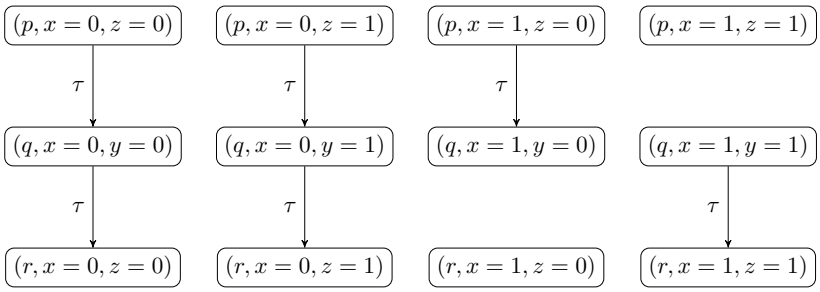
Der Serviceautomat A', der aus A durch Elimination *einer Variable x von q hervorgeht, ist der Serviceautomat, der aus A entsteht, indem*

- *x aus der Menge der Variablen von q entfernt wird,*
- *der Guard jeder eingehenden Kante e von q durch seine Einschränkung auf die Variablenmenge $var(e) \setminus \{x^{new}\}$ ersetzt wird und*
- *der Guard G jeder ausgehenden Kante e von q durch seine Einschränkung auf die Variablenmenge $var(e) \setminus \{x\}$ ersetzt wird.*

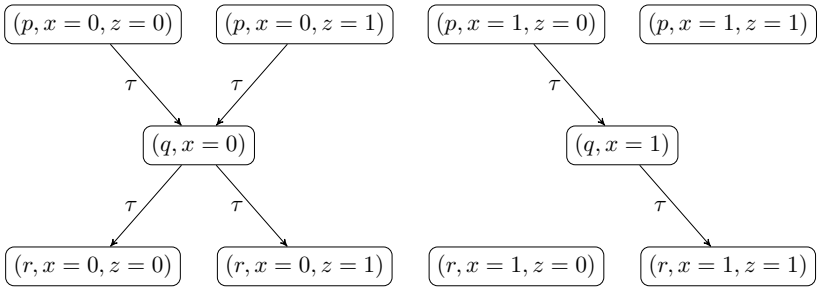
Auf Kanten, welche eine Schleife bilden, werden dabei beide Einschränkungen nacheinander angewendet.

Die Einschränkung der Guards können wir syntaktisch durch existentielle Quantifizierung der eliminierten Variable ausdrücken.

Das Eliminieren einer Variablen entspricht im Transitionssystem des Serviceautomaten dem Verschmelzen vieler Zustandsmengen zu jeweils einem neuen Zustand. Es werden



(a) vor Elimination von y in q



(b) nach Elimination von y in q

Abbildung 2.12.: Transitionssystem (Ausschnitt) des Serviceautomaten in Abb. 2.12

dabei alle Zustände, deren Variablenbelegung sich nur in der eliminierten Variablen unterscheidet, zu jeweils einem Zustand verschmolzen.

Beispiel Im Beispiel aus Abb. 2.11 werden die Zustände $(q, x = 0, y = 0)$ und $(q, x = 0, y = 1)$ zu dem Zustand $(q, x = 0)$ verschmolzen (Abb. 2.12). Analog dazu werden die Zustände $(q, x = 1, y = 0)$ und $(q, x = 1, y = 1)$ zu dem Zustand $(q, x = 1)$ verschmolzen (Abb. 2.12).

Korollar 2.10 *Sei x eine Variable eines Knotens q eines Serviceautomaten $A = (Q, C, var, q_0, E, \Omega)$. A' sei der Serviceautomat, der aus A durch Elimination von x aus q entsteht. Zu jeder Belegung β von $var(q) \setminus \{x\}$ sei Z_β die Menge der Instanzen von q, deren Variablenbelegung auf allen Variablen außer x mit β übereinstimmt.*

Dann ist die Entfaltung von A' isomorph zu dem Zustandsautomaten, der aus der Entfaltung von A durch Verschmelzen jeder Menge Z_β von Zuständen zu jeweils einem Zustand entsteht.

Jeder Schritt im Transitionssystem des Zustandsautomaten entspricht einer Belegung, die den Guard einer dem Knoten angrenzenden Kante erfüllt.

Ein Schritt, der durch Einschränkung seiner Belegung auf die Variablenmenge eines Knotens, aus dem eine Variable eliminiert wurde, entsteht, führt im neuen Transitionssystem zu dem Zustand, der ebenfalls durch Einschränkung seiner Belegung aus dem Zielzustand des ursprünglichen Schrittes entsteht.

Beispiel Sei e die Kante von q nach r.
Aus dem Schritt $(q, x = 0, y = 0) \xrightarrow{e,\gamma} (r, x = 0, z = 0)$ mit

$$\gamma : x \mapsto 0, y \mapsto 0, x^n \mapsto 0, z^n \mapsto 0$$

wird nach Eliminierung von y durch Einschränkung auf die Variablenmenge $\{x, x^n, z^n\}$ der Schritt $(q, x = 0) \xrightarrow{e,\gamma'} (r, x = 0, z = 0)$ mit

$$\gamma' : x \mapsto 0, x^n \mapsto 0, z^n \mapsto 0$$

3. Stand der Technik

Wir geben einen kurzen Überblick über formale Modelle von Services, die Daten berücksichtigen.

Mit den in dieser Arbeit verwendeten Serviceautomaten vergleichbare Formalismen findet man in [84, 61, 14]. Diese Arbeiten erweitern endliche Automaten um Variablen oder Parametervektoren. Die Beschriftung einer Kante eines Automaten besteht aus einem Ereignis, einem Guard und einer Funktion, welche den Variablen neue Werte zuweist. Die Variablen sind allerdings im Unterschied zu unserem Modell global und nicht an einzelne Knoten gebunden. Bei der Analyse der mit den Formalismen beschriebenen Systeme wird meist auf Werkzeuge zur Verifikation endlicher Automaten zurückgegriffen. Dabei werden z. B. BDDs [13] eingesetzt.

Services, die Daten untereinander austauschen, die aus einer unendlich großen Domäne stammen, haben nicht nur unendlich viele Zustände, sondern auch ein unendlich großes Alphabet. Für Systeme dieser Art benötigen wir geeignete Analysemethoden.

Es gibt einige durch Variablen erweiterte endliche Automaten die Sprachen erkennen, die über einem unendlich großen Alphabet gebildet werden. Dazu gehören z. B. *Guarded Variable Automata* [9], Finite Memory Automata [42, 68], Fresh Variable Automata [8], *Data Automata* [11] und *Variable Automata* [29]. Nicht alle von diesen sind darauf ausgelegt oder dazu geeignet, Transitionssysteme von Services zu beschreiben. Die Kriterien dafür, wann ein Wort von einem Automaten akzeptiert wird, sind zum Teil recht komplex. Guarded Variable Automata und Data Automata erkennen Worte, deren Buchstaben paarweise in Gleichheits- oder Ungleichheitsbeziehung zueinander stehen. In dieser Beziehung sind sie den später in Abschnitt 5.2 definierten Identitätsautomaten ähnlich.

Hennessy et al. [34, 51] stellen *symbolische Transitionssysteme* (STS) vor, eine an Prozessalgebra angelehnte graphbasierte Repräsentation eines Transitionssystems, die Terme verwendet, um Werte zu repräsentieren. Jeder Knoten eines STS hat wie in unserem Formalismus seine eigene Menge von Variablen. Mit jeder Aktion des Transitionssystems kann ein Wert aus einer unendlich großen Menge assoziiert sein. Symbolische Transitionssysteme ähnlich denen in [34, 51] werden unter anderem zum Analysieren von Web Services benutzt [71]. Mit diesen werden z. B. Eigenschaften wie *boundedness* [59] oder Simulationsrelationen zwischen Systemen berechnet [34, 7, 98, 71].

Für Petrinetze gibt es verschiedene Erweiterungen, mit denen Daten dargestellt werden können. Ein Zustand eines Petrinetzes wird durch *Marken* definiert, welche sich auf den *Plätzen* des Petrinetzes befinden. Daten werden in Petrinetzen meist durch farbige Marken dargestellt. Für technische Details sei hier auf die Standardliteratur verwiesen [97, 38]. Für einige Petrinetzvarianten existieren kompakte Repräsentationen des Zustandsraumes, mit denen Eigenschaften wie z. B. die Erreichbarkeit von Markierungen

analysiert werden kann. Viele Techniken zur Erreichbarkeitsanalyse nutzten Symmetriebeziehungen zwischen Farben aus [39, 37, 30].

Well-Formed Coloured Nets (WFN) [15] sind eine Klasse gefärbter Petrinetze, die nur bestimmte Guards und Funktionen verwendet. Jede Farbe, die eine Marke eines WFN annehmen kann, ist Element einer von endlich vielen Domänen $D_1, \ldots, D_n$. Der *symbolische Erreichbarkeitsgraph* (SRG) eines WFN ist eine kompakte Repräsentation des Zustandsraumes, die Symmetrien zwischen Farben der gleichen Domäne ausnutzt. Einen Spezialfall dieses SRG werden wir für eine Serviceklasse in Abschnitt 13.1 definieren.

Van der Werf [32] entwirft eine Petrinetzklasse, um Datenbanktransitionen zu modellieren und zu analysieren. Für diese definiert er eine Zustandsraumrepräsentation, mit der die Erreichbarkeit von Markierungen entschieden werden kann.

Lazić et al. [50] stellen *Data Nets*, eine Verallgemeinerung von Petrinetzen vor, deren Marken Werte aus unendlich großen linear geordneten Domänen annehmen können. Die Autoren zeigen, dass Eigenschaften wie Überdeckbarkeit und Terminierung für Data Nets entscheidbar sind.

Schmidt [82, 81] definiert eine Klasse algebraischer Petrinetze ohne Guards. Für ein algebraisches Petrinetz definiert er eine kompakte Repräsentation des Zustandsraumes, den *parametrisierten Erreichbarkleitsgraphen*. Dieser ist unter bestimmten Voraussetzungen endlich. Mit Hilfe des parametrisierten Erreichbarkeitsgraphen kann entschieden werden, ob bestimmte Markierungen des Petrinetzes erreichbar sind.

Pommerau et al. [73] definieren *Petri nets with counters* (PNZ), eine Klasse von place/transition Netzen erweitert um einen globalen Vektor von integer-Variablen, auf denen lineare Transformationen ausgeführt werden können. Jede Transition eines PNZ besitzt als zusätzliche Anschrift eine *linear condition* (z. B. $x < y \wedge x + 2y = 3z$) und ein *linear update* (z. B. $x := 2x + y$). Für ein PNZ definieren die Autoren eine kompakte Repräsentation des Erreichbarkeitsgraphen. Diese erhält die exakte Semantik des PNZ, d. h. Menge der erreichbaren Zustände und die Menge der ausführbaren Transitionssequenzen.

4. Zusammenfassung

In diesem Abschnitt haben wir ein formales Servicemodell vorgestellt, in welchem die Verarbeitung von Daten durch einen Service beschrieben werden kann.

Das Servicemodell erweitert einen endlichen Automaten um Variablen und Prädikate. Dies ermöglicht die Beschreibung eines Services mit unendlich vielen Zuständen. Wir haben bewusst offen gelassen, wie die Prädikate formal dargestellt werden. Die Prädikate werden formal zunächst als (im allgemeinen unendlich große) Mengen von Belegungen definiert. Wir legen hier noch keine endliche Darstellung der Prädikate in einem bestimmten Formalismus fest. Dies erleichtert es uns später, für jede Serviceklasse und Analysemethode eine für diese geeignete Darstellung zu wählen.

Im nun folgenden Teil 2 der Arbeit stehen semantische Aspekte des Verhaltens im Vordergrund, für welche die gewählte Darstellung der Prädikate keine entscheidende Rolle spielt. Die konkrete Darstellung der Prädikate des Serviceautomaten wird erst in Teil 3 der Arbeit relevant. Die dort vorgestellten Algorithmen benötigen eine konkrete endliche Darstellung, um Mengen von Zuständen mit rechnergestützten Mitteln berechnen und endlich darstellen zu können.

Wir haben grundlegende Operationen und Konzepte für Serviceautomaten definiert.

Die wichtigste Operation ist die *Komposition* von Services. Wir komponieren Services formal durch Bildung des Kreuzproduktes. Im Gegensatz zu dem in [96, 57] verwendeten Servicemodell kommunizieren unsere Serviceautomaten nicht asynchron, sondern synchron. Asynchrone Kommunikation können wir auf synchrone Kommunikation zurückführen. Da in Teil 3 der Arbeit nur azyklische Serviceautomaten betrachtet werden, ist die maximale Anzahl von Nachrichten im Nachrichtenpuffer beschränkt. Eine Sonderbehandlung der asynchronen Kommunikation, um die Anzahl der Nachrichten im Nachrichtenpuffer zu beschränken und damit die endliche Darstellbarkeit des Zustandsraumes zu gewährleisten, ist daher nicht erforderlich.

Weiterhin haben wir Operationen zum Entfernen und Verschmelzen von Knoten bzw. zum Eliminieren von Variablen definiert. Diese Operationen werden wir in späteren Kapiteln benutzen.

Ein wichtiges Konzept zur Analyse von Kompositionen zweier Services ist die *Wissensfunktion*. Die Wissensfunktion ist ein nützliches Konzept, um den Zusammenhang zwischen den Zuständen einzelner Serviceautomaten und Zuständen von Kompositionen von Serviceautomaten zu beschreiben. Sie spielt eine zentrale Rolle beim Verschmelzen von Knoten und bei der Synthese von Partnern. Die Wissensfunktion werden wir in Teil 3 dieser Arbeit auf Klassen von (potenziell unendlich vielen) Zuständen von Serviceautomaten erweitern. Eine solche Erweiterung auf unendlich viele Zustände ist bisher nur möglich für die Serviceklasse der Identitätsautomaten, die wir in Abschnitt 5.2 vorstellen werden.

Im folgenden Teil 2 definieren wir den Begriff des *Partners* eines Service und untersuchen, wann ein Service *bedienbar* ist.

Teil II.

Partner eines Service

5. Partner und Bedienbarkeit eines Service

Services werden mit dem Ziel entworfen, dass sie eine bestimmte *Aufgabe* erfüllen. Die Aufgabe kann z. B. darin bestehen, einen Geschäftsprozess erfolgreich abzuschließen. Ein Service ist eingebunden in ein verteiltes System, das aus mehreren Services besteht. Zum Erfüllen der Aufgabe sind die Services aufeinander angewiesen. Kein Service allein kann die Aufgabe erfüllen, sondern muss dazu mit mindestens einem anderen Service interagieren.

In dieser Arbeit betrachten wir die Interaktion von jeweils genau zwei Services miteinander. Damit zwei Services erfolgreich miteinander zusammenarbeiten können, muss ihr Verhalten miteinander *kompatibel* sein. Das Verhalten zweier Services ist kompatibel, wenn die Komposition der Services ein *Kompatibilitätskriterium*, eine zum Erfüllen der Aufgabe notwendige Eigenschaft, erfüllt.

Es gibt viele unterschiedliche Kompatibilitätskriterien. Ein elementares Kriterium ist die *Deadlockfreiheit*. Ein Deadlock ist ein Systemzustand, in dem jeder der beteiligten Services blockiert ist und nicht mit der Ausführung fortfahren kann. Deadlocks treten häufig in verteilten Systemen auf und verhindern, dass die Services ihre gemeinsame Aufgabe erfüllen können.

Definition 5.1 (Deadlock) *Ein Zustand eines Serviceautomaten A heißt* Deadlock*, wenn er keinen Nachfolgerzustand hat. Ein Serviceautomat A heißt* deadlockfrei*, wenn kein erreichbarer Zustand ein Deadlock ist.*

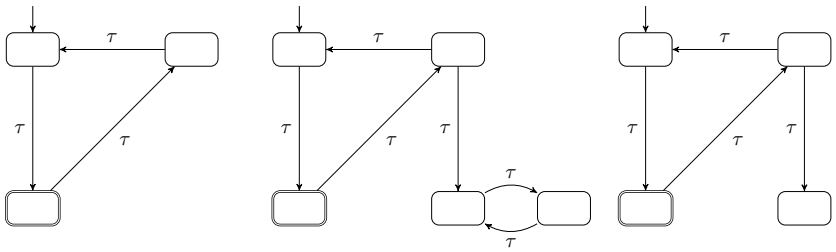
Ein stärkeres Kompatibilitätskriterium ist *schwache Terminierung*. Ein Serviceautomat terminiert schwach, wenn von jedem erreichbaren Zustand ein Endzustand erreichbar ist. Das Erfüllen einer Aufgabe durch die Services können wir in vielen Fällen durch das Erreichen ihrer jeweiligen Endzustände darstellen.

Definition 5.2 (Schwache Terminierung) *Ein Serviceautomat A heißt* schwach terminierend*, wenn es für jeden erreichbaren Zustand z einen Endzustand z' gibt, so dass $z \xrightarrow{*}_A z'$.*

Ein Serviceautomat, in dem ein Deadlock erreichbar ist, terminiert nicht schwach.

Neben diesen beiden elementaren Kompatibilitätskriterien gibt es weitere Kompatibilitätskriterien. Varianten der beiden genannten Kompatibilitätskriterien sind z. B. *responsiveness* [56] und *starke Terminierung* [95]. Komplexere Kompatibilitätskriterien können beispielsweise durch temporallogische Formeln spezifiziert werden [72, 49].

Zwei Services heißen *Partner*, wenn ihre Komposition das Kompatibilitätskriterium erfüllt. Zu jedem Kompatibilitätskriterium ergibt sich eine Partnerrelation. In dieser Arbeit betrachten wir ausschließlich das Kompatibilitätskriterium schwache Terminierung.



(a) A_1 (b) A_2 (c) A_3

Abbildung 5.1.: Kompatibilitätskriterien. A_1 ist deadlockfrei und terminiert schwach. A_2 ist deadlockfrei aber terminiert nicht schwach. A_3 ist nicht deadlockfrei und terminiert nicht schwach.

Daher definieren wir den Begriff des Partners speziell für dieses Kompatibilitätskriterium.

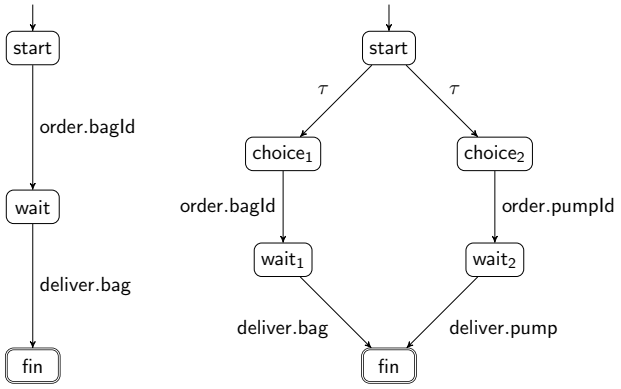
Definition 5.3 (Partner) *Zwei Serviceautomaten A und B heißen* Partner*, wenn $A \times B$ schwach terminiert.* Partners(B) *bezeichne die Menge der Partner von B.*

Ein Partner entspricht einem Regler (*controller*) in der Kontrolltheorie [75]. In [83, 57, 96] wird auch die Bezeichnung *Strategie* für Partner verwendet. Der Begriff des Partners wird in [57] für asynchron über einen unbeschränkten Nachrichtenpuffer kommunizierende Services definiert.

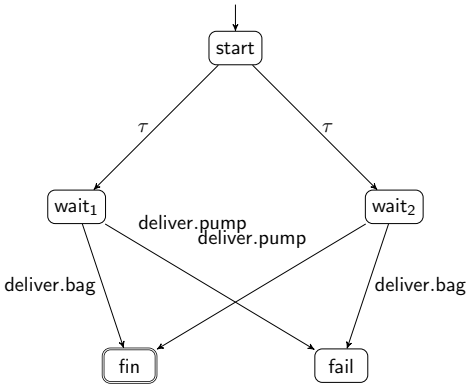
Für asynchron kommunizierende Serviceautomaten können wir einen analogen Partnerbegriff bilden. Zwei asynchron kommunizierende Serviceautomaten A und B sind Partner, wenn $A \otimes P \otimes B$ schwach terminiert. Der Serviceautomat P bezeichnet dabei die Komposition aller Nachrichtenpuffer (siehe Abschnitt 2.3.1), über die A und B kommunizieren. Alle Theoreme dieser Arbeit beziehen sich stets auf Partner nach Definition 5.3.

Beispiel Wir illustrieren den Partnerbegriff anhand eines Beispiels aus dem Katastrophenschutz. Abb. 5.2 zeigt einen Mitarbeiter des Katastrophenschutzes Worker_1. Der Mitarbeiter benötigt Sandsäcke, um einen Damm gegen eine bevorstehende Flut zu errichten. Die Sandsäcke hat er nicht vorrätig, sondern muss sie bei einem Vorratslager des Katastrophenschutzes anfordern. Zu diesem Zweck hat der Mitarbeiter einen Kanal order, über den Bestellungen für verschiedene Ausrüstungsgegenstände verschickt werden können. Das Bestellen eines Sandsacks entspricht dem Senden der Bestellnummer des Sandsackes bagId auf Kanal order. Über den Kanal deliver kann der Arbeiter gelieferte Gegenstände entgegennehmen.

Das Vorratslager Supply_1 nimmt Bestellungen für Sandsäcke entgegen. Wenn Worker_1 auf Kanal order die Bestellnummer bagId des Sandsacks schickt, empfängt Supply_1 diese Nachricht und schickt über den Kanal deliver einen Sandsack zurück. Diesen empfängt

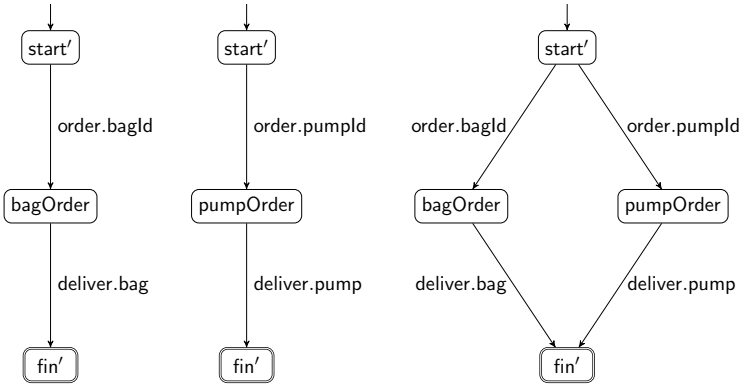


(a) Worker$_1$ (b) Worker$_2$



(c) Worker$_3$

Abbildung 5.2.: Mitarbeiter des Katastrophenschutzes



(a) Supply$_1$ (b) Supply$_2$ (c) Supply$_3$

Abbildung 5.3.: Vorratslager des Katastrophenschutzes

	Supply_1	Supply_2	Supply_3	Supply_4
Worker_1	✓	-	✓	✓
Worker_2	-	-	✓	✓
Worker_3	-	-	-	-
Worker_4	-	-	-	✓

Tabelle 5.1.: Partner der Katastrophenschutzmitarbeiter

Worker_1 auf Kanal deliver und schließt seine Arbeit am Damm ab. Danach sind beide Serviceautomaten in ihrem Endzustand. Damit sind Worker_1 und Supply_1 Partner.

Dagegen sind Worker_1 und Supply_2 keine Partner. Supply_2 hat ausschließlich Pumpen vorrätig. Supply_2 kann im Gegensatz zu Supply_1 keine Bestellungen von Sandsäcken entgegen nehmen. In Zustand $(\mathsf{start}, \mathsf{start}')$ des Kreuzproduktes kann weder die Aktion order.bagId noch die Aktion order.pumpId ausgeführt werden. Der Zustand ist ein Deadlock. Worker_1 und Supply_2 erreichen daher ihren jeweiligen Endzustand nicht.

Supply_3 hat sowohl Sandsäcke als auch Pumpen vorrätig und ist daher ein Partner von Worker_1.

Der Katastrophenschutzmitarbeiter Worker_2 kann zwei verschiedene Tätigkeiten ausführen: Dämme errichten oder Wassereinbrüche abpumpen. Der Mitarbeiter entscheidet sich nichtdeterministisch für eine der beiden Tätigkeiten und bestellt entweder einen Sandsack oder eine Pumpe. Weder Supply_1 noch Supply_2 sind Partner von Worker_2. Im Kreuzprodukt von Worker_2 und Supply_1 wird der Deadlock $(\mathsf{choice}_2, \mathsf{start}')$ erreicht, wenn Worker_2 eine Pumpe bestellt. Eine analoge Situation ergibt sich für die Komposition von Worker_2 und Supply_2.

Supply_3 ist ein Partner von Worker_2, da sowohl Bestellungen für Sandsäcke als auch für Pumpen entgegengenommen werden können.

Worker_3 hat keinen Partner. Der Serviceautomat entscheidet sich zunächst nichtdeterministisch dazu, in einen der beiden Zustände wait_1 oder wait_2 zu gehen. In Zustand wait_1 benötigt Worker_3 einen Sandsack, um in seinen Endzustand zu gelangen. In Zustand wait_2 benötigt Worker_3 eine Pumpe, um in seinen Endzustand zu gelangen. Falls der Serviceautomat einen anderen als den benötigten Gegenstand erhält, geht er in einen Deadlock.

Die Entscheidung, ob Worker_3 in den Zustand wait_1 oder wait_2 geht, wird nicht kommuniziert. Daher kann ein potentieller Partner nicht wissen, welcher Gegenstand geliefert werden muss, damit Worker_3 garantiert in seinen Endzustand gelangt. Daher hat Worker_3 keinen Partner. Wir sagen, Worker_3 ist nicht *bedienbar*.

Ein Service heißt *bedienbar*, wenn er mindestens einen Partner hat.

Definition 5.4 (Bedienbarkeit) *Ein Serviceautomat A heißt* bedienbar*, wenn es mindestens einen Partner P von A gibt. A heißt* bedienbar *innerhalb einer Menge von Serviceautomaten $\mathcal{S}$, wenn es einen Partner P von A mit $P \in \mathcal{S}$ gibt.*

In dieser Arbeit betrachten wir ausschließlich die Bedienbarkeit eines Service für das Kompatibilitätskriterium schwache Terminierung. Für das Kompatibilitätskriteri-

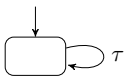
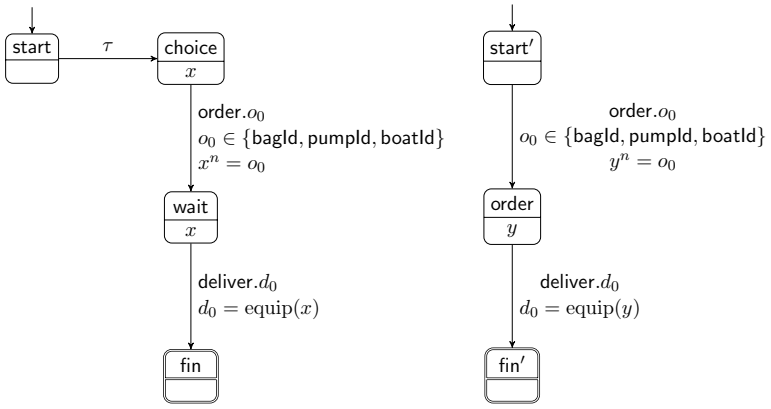


Abbildung 5.4.: Serviceautomat InternalLoop



(a) Worker$_4$ (b) Supply$_4$

Abbildung 5.5.: Serviceautomaten mit Variablen

um Deadlockfreiheit ist der Serviceautomat InternalLoop in Abb. 5.4 ein Partner jedes Serviceautomaten. Die Komposition erreicht nie einen Deadlock, da InternalLoop immer einen τ-Schritt ausführen kann. Ein Service ist für dieses Kompatibilitätskriterium daher immer trivial bedienbar.

Die Menge der möglichen Partner schränken wir in der Regel auf Serviceautomaten einer bestimmten Klasse ein.

Nicht bedienbare Services sind inhärent fehlerhaft, da sie niemals ihre Aufgabe erfüllen können. Ein Ziel dieser Arbeit ist, entscheiden zu können, ob ein Service bedienbar ist. Zu wissen, ob ein Service bedienbar ist, ist für den Entwickler des Services ein großer Vorteil. Fehler im Design des Service können damit frühzeitig erkannt und behoben werden. Ein Entscheidungsverfahren für Bedienbarkeit kann dem Entwickler viele aufwändige Tests ersparen und verhindern, dass grundlegende Fehler erst während des Betriebs gefunden werden.

Ziel dieser Arbeit ist, ein Entscheidungsverfahren für die Bedienbarkeit eines Service zu finden. Wir betrachten dabei verschiedene Klassen von Serviceautomaten. Der Schwerpunkt liegt dabei auf Serviceautomaten mit unendlich vielen Zuständen. Diese eignen sich gut, um die Verarbeitung von Daten in Services formal zu beschreiben.

Um einen Serviceautomaten mit unendlich vielen Zuständen in unserem Formalismus endlich darzustellen, benötigen wir Variablen.

Beispiel Der Serviceautomat Worker$_4$ in Abb. 5.5 hat eine Variable x und eine Funktion equip. Die Domäne von x ist formal das Universum $\mathcal{U}$, welches hier sowohl Ausrüstungsgegenstände als auch deren Bestellnummern umfasst. In diesem konkreten Fall kann x

Werte aus der Menge der Bestellnummern {bagId, pumpId, boatId} annehmen. Die Funktion equip ordnet jeder Bestellnummer seinen Gegenstand zu.

Worker_4 verhält sich ähnlich wie Worker_2. Worker_4 wählt zufällig eine Bestellnummer eines Ausrüstungsgegenstandes aus, sendet diese auf dem Kanal order und speichert sie in der Variablen x. Im Knoten wait wartet Worker_4 dann auf das Eintreffen des bestellten Ausrüstungsgegenstandes. Nur wenn genau dieser Gegenstand vom Kanal deliver empfangen wird, kann Worker_4 in seinen Endzustand gelangen. Der Serviceautomat Supply_4 ist ein Partner von Worker_4.

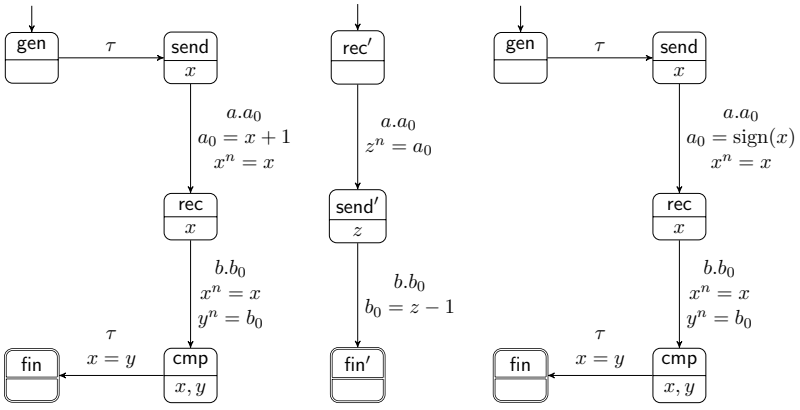
Für ein Servicemodell für Services mit endlich vielen Zuständen existiert bereits ein Entscheidungsverfahren für Bedienbarkeit [96]. Dieses Modell abstrahiert von konkreten Daten. Nachrichten, die auf einem Kanal gesendet oder empfangen werden, werden nicht nach ihrem Inhalt unterschieden. Das Verhalten eines Service kann daher in der Regel nur vergleichsweise grob wiedergegeben werden. Eine Entscheidung eines Service darüber, wie eine Nachricht verarbeitet werden soll, die in der realen Welt vollständig vom Inhalt einer Nachricht bestimmt wird, wird in diesem Modell meist als nichtdeterministische Entscheidung dargestellt. Auf diese Weise wird jeder Ausgang einer Entscheidung abgedeckt. Diese Darstellung ist hinreichend genau, solange zwischen den Nachrichten keine inhaltlichen Abhängigkeiten bestehen. Die Zuordnung zwischen Bestellnummern und Ausrüstungsgegenständen wie in unserem Beispiel geht in einer solchen Darstellung verloren. Zwischen der Bestellung eines Sandsackes und einer Pumpe wird nicht unterschieden. Diese von Daten abstrahierende Darstellung ist daher nicht fein genug, um ableiten zu können, dass bei Lieferung eines anderen als des bestellten Ausrüstungsgegenstandes ein Deadlock erreicht wird.

In dieser Arbeit wollen wir ein Entscheidungsverfahren entwickeln für ein Modell von Serviceautomaten, welches Daten explizit berücksichtigt. Die Daten können im allgemeinen aus einer unendlich großen Domäne stammen. Die von uns betrachteten Serviceautomaten haben daher im allgemeinen unendlich viele Zustände.

Serviceautomaten mit endlich vielen Zuständen können mit dem in [96] beschriebenen Servicemodell analysiert werden. Dazu wird der Serviceautomat einfach durch Entfaltung in einen äquivalenten endlichen Automaten umgewandelt. Die Entfaltung hat für jede mögliche Nachricht einen eigenen Kanal. Auf diese Weise werden die verschiedenen Ausprägungen der über einen Kanal kommunizierten Daten unterscheidbar und innerhalb des Formalismus analysierbar.

Für Services mit unendlich vielen Zuständen ist eine Reduktion dieser Art nicht möglich, da die Entfaltung dann selbst unendlich groß ist. Um Services mit unendlich vielen Zuständen analysieren zu können, benötigen wir Verfahren, die explizit Bezug auf die verwendeten Variablen, Funktionen und Relationen nehmen. Serviceautomaten, deren Variablen (im Unterschied etwa zu der Variable von Worker_4) tatsächlich unendlich viele verschiedene Werte annehmen können, sind im allgemeinen wesentlich schwieriger zu analysieren, insbesondere im Zusammenspiel mit Funktionen. Ob ein Service bedienbar ist oder nicht, hängt empfindlich von den benutzten Funktionen ab.

Wir betrachten dazu den Serviceautomaten AddFunc in Abb. 5.6. AddFunc wählt nichtdeterministisch eine ganze Zahl n und kommuniziert die Zahl $n + 1$ auf dem Kanal a.



(a) AddFunc (b) SubFunc (c) SignFunc

Abbildung 5.6.: Serviceautomaten mit unendlich vielen Zuständen

Danach empfängt AddFunc eine zweite Zahl m auf Kanal b und vergleicht diese mit n. Ist die empfangene Zahl m gleich n, geht AddFunc in seinen Endzustand. Andernfalls verbleibt AddFunc in einem Deadlock.

Der Serviceautomat SubFunc ist ein Partner von AddFunc. SubFunc zieht von der Zahl, die er auf Kanal a erhält, eins ab und sendet diese auf Kanal b. Es ist intuitiv klar, dass jeder Partner von AddFunc sich so ähnlich verhält wie SubFunc, d. h. er sendet auf Kanal b eine Zahl, die um eins kleiner ist als die Zahl, die er auf a empfangen hat. Jeder Partner von SubFunc hat unendlich viele Zustände. Daher ist SubFunc innerhalb der Menge der Serviceautomaten ohne Variablen und mit endlich vielen Knoten nicht bedienbar.

Ersetzt man die umkehrbare Additionsfunktion in AddFunc durch eine nicht umkehrbare Funktion, ist der Serviceautomat nicht mehr bedienbar. Der Serviceautomat SignFunc ist mit AddFunc bis auf die benutzte Funktion identisch. SignFunc benutzt statt der Additionsfunktion die Signumfunktion, d. h. er sendet auf a das Vorzeichen der nichtdeterministisch gewählten Zahl. SignFunc ist nicht bedienbar, da aus dem Vorzeichen der vom Serviceautomaten gewählten Zahl die ursprüngliche Zahl nicht rekonstruiert werden kann.

Der Rest dieses Teils ist wie folgt gegliedert: Wir zeigen zunächst, dass Bedienbarkeit im allgemeinen nicht entscheidbar ist (Abschnitt 5.1).

Wir betrachten dann in Abschnitt 5.2 Serviceautomaten, die nur Variablen, aber keine Funktionen oder Konstanten verwenden und zeigen, dass mindestens die Gleichheits- und die Ungleichheitsrelation notwendig sind, um Partner für diese Serviceautomaten darstellen zu können. Einen Serviceautomaten, der Werte nur bezüglich paarweiser Gleichheit oder Ungleichheit unterscheidet, nennen wir *Identitätsautomat*.

In den Abschnitten 5.3 und 6.1 zeigen wir, dass unter bestimmten Voraussetzungen Zustände von Partnern verschmolzen werden oder Variablen von Partner eliminiert werden können.

Wir zeigen in Abschnitt 6.2, dass Partner von Identitätsautomaten im allgemeinen

unendlich viele Variablen benötigen und daher nicht endlich darstellbar sind.

5.1. Unentscheidbarkeit der Bedienbarkeit

In diesem Abschnitt zeigen wir, dass die Bedienbarkeit eines Serviceautomaten im allgemeinen unentscheidbar ist.

Für ungefärbte offene Petrinetze ist Bedienbarkeit unentscheidbar [62], falls die Anzahl der Marken auf den Plätzen des offenen Petrinetzes nicht beschränkt ist. Petrinetze mit unbeschränkten Plätzen haben im allgemeinen unendlich viele Zustände. Die Bedienbarkeit eines Services, der durch einen asynchron kommunizierenden endlichen Automaten beschrieben werden kann, ist dagegen entscheidbar unter der Nebenbedingung, dass die Anzahl der Nachrichten in den Nachrichtenpuffern beschränkt ist [96].

Für unser Servicemodell wird die Unentscheidbarkeit der Bedienbarkeit nicht durch der Anzahl die Nachrichten bedingt, sondern ergibt sich daraus, dass jede einzelne Nachricht einen von unendlich vielen Werten annehmen kann.

Wir zeigen, dass wir eine *Zählermaschine* als Spezialfall eines Serviceautomaten ausdrücken können. Zählermaschinen sind turingmächtig [66].

Eine Zählermaschine ist eine Registermaschine mit einer endlichen Anzahl von Registern („Zählern“), die jeweils nicht-negative ganze Zahlen speichern. Der Befehlssatz einer Zählermaschine besteht aus den zwei Befehlen INC und JZDEC. INC erhöht den Inhalt eines Registers um 1, JZDEC vermindert ihn um 1 oder führt einen Sprung aus, falls der Inhalt des Registers gleich 0 ist.

Definition 5.5 (Zählermaschine) *Eine n-Zählermaschine ZM besteht aus einer endlichen Menge von Zuständen Z, einem Anfangszustand $z_0 \in Z$, einem Endzustand $z_f \in Z$, n Zählern $c_1, \ldots, c_n$ und einer endlichen Menge von Anweisungen der Form*

$$z\text{:}\ \mathrm{INC}(c_i, z)$$
bzw. $$z\text{:}\ \mathrm{JZDEC}(c_i, z', z'')$$

wobei $z \in Z \setminus \{z_f\}, z', z'' \in Z$.

Die Semantik der Befehle können wir in Pseudocode wie folgt ausdrücken:

$\mathrm{INC}(c_i, z)$: $\quad c_i := c_i + 1$; goto z';
$\mathrm{JZDEC}(c_i, z', z'')$: $\quad$ if $c_i = 0$ then goto z' else $c_i := c_i - 1$; goto z'';

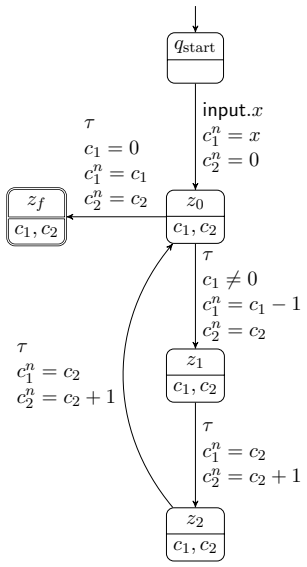
Die *Eingabe* einer Zählermaschine ZM ist eine Zahl $x \in \mathbb{N}$. In der Anfangskonfiguration der Zählermaschine ZM ist das erste Register mit x initialisiert. Alle anderen Register sind mit 0 initialisiert. Eine Zählermaschine *akzeptiert* die Eingabe x, wenn die Zählermaschine von der Anfangskonfiguration aus ihren Endzustand erreichen kann. Es ist unentscheidbar, ob eine Zählermaschine eine Eingabe akzeptiert [66].

Theorem 5.1 *Es ist unentscheidbar, ob eine 2-Zählermaschine eine Eingabe $x \in \mathbb{N}$ akzeptiert.*

Wir reduzieren das Problem, zu entscheiden, ob eine Zählermaschine ZM eine Eingabe akzeptiert, auf das Problem, zu entscheiden, ob zwei Serviceautomaten Partner

Programm	
z_0	JZDEC(c_1, z_f, z_1)
z_1	INC(c_2, z_2)
z_2	INC(c_2, z_0)
z_f	(Accept)

(a) ZM



(b) A_{ZM}

Abbildung 5.7.: Programm einer 2-Zählermaschine und der daraus generierte Serviceautomat

sind. Dazu wandeln wir die Zählermaschine ZM und ihre Eingabe x jeweils in einen Serviceautomaten um:

Für jede Zählermaschine ZM konstruieren wir einen Serviceautomaten A_{ZM}. Die Knotenmenge des Serviceautomaten ist die Zustandsmenge der Zählermaschine erweitert um einen zusätzlichen Knoten q_{start}. A_{ZM} hat einen einzigen Kanal input der Stelligkeit 1, der dazu dient, die Eingabe x zu lesen. q_{start} und z_f sind Anfangs- bzw. Finalknoten von A_{ZM}. Jeder Knoten mit Ausnahme von q_{start} hat die Variablenmenge $c_1, \ldots, c_n$, d. h. eine Variable für jedes Register. q_{start} hat keine Variablen. Von q_{start} geht eine Kante zum Anfangszustand z_0 der Zählermaschine. Diese Kante setzt das erste Register c_1 auf die über den Kanal input kommunizierte Eingabe x und setzt alle anderen Register auf 0. Die Anweisungen der ZM werden folgendermaßen in Kanten übersetzt:

1. Für jede Anweisung der Form z: INC(c_i, z') existiert eine τ-Kante von z zu z' mit dem Guard $c_i^n = c_i + 1 \wedge \bigwedge_{1 \leq j \leq n, i \neq j} c_j^{\text{new}} = c_j$.

2. Für jede Anweisung der Form z: JZDEC(c_i, z', z'') existiert eine τ-Kante von z zu z' mit dem Guard $c_i = 0 \wedge \bigwedge_j c_j^{\text{new}} = c_j$ und eine τ-Kante von z zu z'' mit dem Guard $c_i \neq 0 \wedge c_i^n = c_i - 1 \wedge \bigwedge_{1 \leq j \leq n, i \neq j} c_j^{\text{new}} = c_j$.

Abb. 5.7 illustriert die Transformation beispielhaft für eine Zählermaschine.

Für jede Eingabe $x \in \mathbb{N}$ bezeichne P_x den Serviceautomaten mit zwei Zuständen, der in seinem Anfangszustand die Aktion input.x ausführt und in seinen Endzustand geht. Dann gilt folgendes Lemma:

Lemma 5.2 *Sei ZM eine Zählermaschine, $x \in \mathbb{N}$. Dann sind A_{ZM} und P_x Partner gdw. ZM akzeptiert die Eingabe x.*

Mit Theorem 5.1 folgt, dass im allgemeinen unentscheidbar ist, ob zwei Serviceautomaten füreinander Partner sind. Es folgt weiterhin, dass auch die Existenz eines Partners unentscheidbar ist.

Theorem 5.3 *Es ist unentscheidbar, ob ein Serviceautomat bedienbar ist.*

Beweis Sei ZM eine Zählermaschine und $x \in \mathbb{N}$ beliebig. Sei ZM' die Zählermaschine, die das erste Register durch eine geeignete Folge von Befehlen auf x setzt und sich dann wie ZM verhält. Dann hält ZM genau dann bei Eingabe x, wenn es einen Partner von $A_{ZM'}$ gibt. Daher ist nicht entscheidbar, ob ZM bedienbar ist. □

Im Rest der Arbeit werden wir uns folglich auf Klassen von Serviceautomaten einschränken, die in ihrer Ausdrucksfähigkeit eingeschränkt sind.

In Abschnitt 11 zeigen wir, dass Bedienbarkeit für azyklische *Presburgerautomaten* entscheidbar ist. Presburgerautomaten können Addition, Subtraktion und Nulltest ausdrücken und damit die Befehle INC und JZDEC simulieren.

In Abschnitt 5.2 führen wir *Identitätsautomaten* ein. Ein Identitätsautomat kann nur prüfen, ob je zwei Variablen mit dem gleichen Wert belegt sind. Die Befehle INC und JZDEC können nicht simuliert werden. Ob Bedienbarkeit für zyklische Identitätsautomaten im allgemeinen Fall entscheidbar ist, ist ein offene Frage.

5.2. Minimale Ausdrucksfähigkeit von Partnern

Im vorangegangenen Abschnitt haben wir gezeigt, dass im allgemeinen unentscheidbar ist, ob ein Serviceautomat bedienbar ist. Wenn wir entscheiden wollen, ob zu einem gegebenen Serviceautomaten ein Partner existiert, müssen wir die Menge der betrachteten Serviceautomaten daher notwendigerweise einschränken.

Die Klasse von Serviceautomaten, auf die wir uns einschränken, muss dabei mit bedacht gewählt werden:

Einerseits darf sie nicht zu groß sein, damit die Existenz von Partnern in dieser Klasse noch entscheidbar ist. Andererseits muss die Klasse groß genug sein, damit ein Partner eines Serviceautomaten der Klasse noch mit den in dieser Klasse zur Verfügung stehenden Ausdrucksmitteln darstellbar ist.

Ein Serviceautomat aus einer Klasse $\mathcal{S}$ von Serviceautomaten hat im allgemeinen sowohl innerhalb als auch außerhalb von $\mathcal{S}$ Partner. Bei ungeschickter Wahl von $\mathcal{S}$ ist es möglich, dass jeder Partner eines Serviceautomaten aus $\mathcal{S}$ außerhalb von $\mathcal{S}$ liegt.

In diesem Abschnitt betrachten wir exemplarisch einen Serviceautomaten, der mit einem Minimum an Ausdrucksmitteln darstellbar ist. Diesen Serviceautomaten können wir allein mit Variablen und der Gleichheitsrelation darstellen, ohne Konstanten und Funktionen zu verwenden.

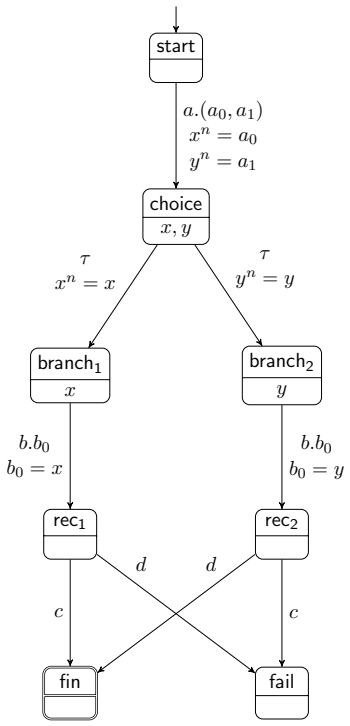


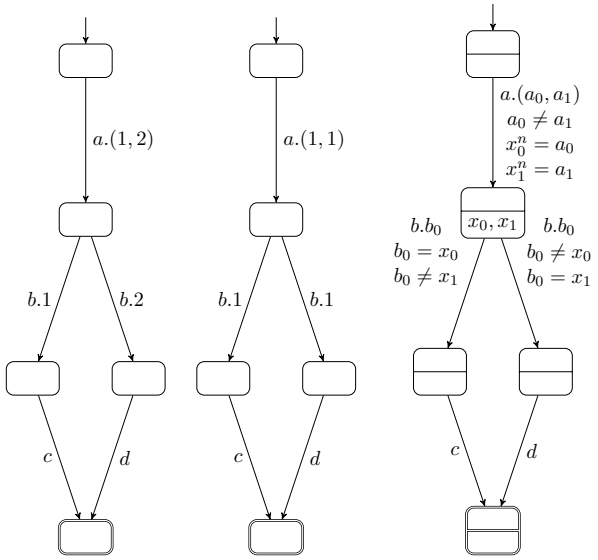
Abbildung 5.8.: Serviceautomat ForceInequal

Wir zeigen, dass zur Darstellung eines Partners dieses Serviceautomaten entweder Konstanten oder die Ungleichrelation notwendig sind. Diese Beobachtung ist insbesondere deshalb wichtig, da die Klasse der *Identitätsautomaten*, d. h. der Serviceautomaten, die nur Variablen, die Gleichheitsrelation und die Ungleichheit benutzen, bereits sehr schwierig zu analysieren ist. In dieser Klasse gibt es, wie wir in Abschnitt 6.2 zeigen werden, Serviceautomaten, deren Partner innerhalb dieser Klasse nicht mehr endlich darstellbar sind. Aufgrund dieses Umstandes können wir daher bereits für sehr elementare Klassen von Serviceautomaten keinen Algorithmus angeben, der zu jedem Serviceautomaten garantiert einen Partner innerhalb derselben Klasse konstruiert.

Wir betrachten den Serviceautomaten ForceInequal in Abb. 5.8. Das Universum von ForceInequal ist die Menge der ganzen Zahlen. Den Serviceautomaten ForceInequal können wir allein unter Verwendung von Variablen und der Gleichheitsrelation darstellen. Die Partner von ForceInequal sind mit dieser Beschränkung nicht darstellbar, wie wir sehen werden:

ForceInequal erhält als Eingabe auf Kanal a zwei Zahlen, die in den Variablen x und y gespeichert werden. Dann wählt der Serviceautomat zufällig eine der beiden Zahlen (x in branch_1, y in branch_2) aus und kommuniziert diese über den Kanal b. In jedem der beiden Zweige führt jeweils eine andere Aktion zum Endknoten (c in branch_1 bzw. d in branch_2).

Der Serviceautomat P_1 ist ein Partner von ForceInequal. P_1 kommuniziert auf Kanal



(a) P_1 (b) P_2 (c) P_3

Abbildung 5.9.: Serviceautomaten mit zu ForceInequal äquivalentem Interface

a zwei voneinander verschiedene Zahlen 1 und 2. Je nachdem, welche der beiden Zahlen ForceInequal auswählt und anschließend auf b kommuniziert, führt P_1 entweder die Aktion c oder d aus. Auf beiden Wegen gelangt ForceInequal in seinen Endzustand.

Der Serviceautomat P_2 ist kein Partner von ForceInequal. P_2 kommuniziert die gleiche Zahl zwei mal an ForceInequal. Anhand der von ForceInequal über den Kanal b kommunizierten Nachricht kann P_2 nicht entscheiden, ob ForceInequal in Zustand rec_1 oder rec_2 ist, da diese Nachrichten in beiden Fällen identisch sind. Daher kann P_2 anhand der Nachrichten nicht entscheiden, ob die Aktion c oder d ausgeführt werden muss, um ForceInequal in seinen Endzustand zu bringen.

Verallgemeinernd können wir schlussfolgern: Jeder Partner von ForceInequal muss auf Kanal a zwei voneinander verschiedene Zahlen kommunizieren. Daher ist auch der Serviceautomat P_3, welcher beliebige voneinander verschiedene Zahlen auf a kommuniziert, ein Partner von ForceInequal.

Der Serviceautomat P_3 ist ein *Identitätsautomat*. Je zwei Variablen eines Guards eines Identitätsautomaten stehen jeweils in einer Gleichheits- oder Ungleichheitsbeziehung zueinander.

Formal definieren wir einen Identitätsautomaten, ohne uns auf eine spezielle syntaktische Darstellung der Guards festzulegen. Wir definieren die Prädikate statt dessen direkt als Mengen von Belegungen. Dieser Ansatz hat den Vorteil, dass wir später in Abschnitt 13 Äquivalenzklassen von Belegungen und Zuständen eines Identitätsautomaten bilden können, ohne auf die konkrete Darstellung der Äquivalenzklassen einzugehen.

Jedes Prädikat eines Identitätsautomaten ist *identitätsbewahrend*, d. h. jedes Paar von Variablen ist in jeder Belegung entweder mit dem gleichen Wert belegt oder mit

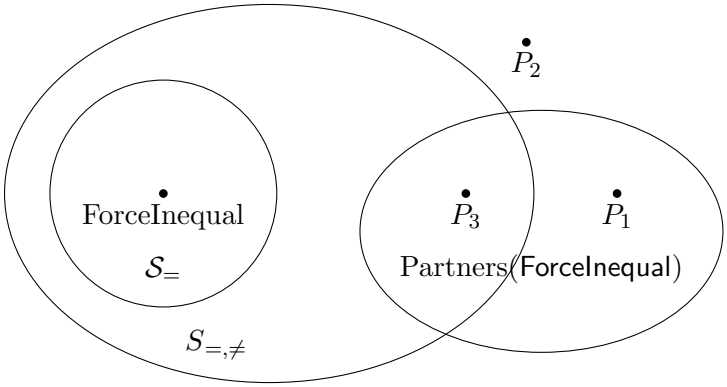


Abbildung 5.10.: Serviceklassen und Partnermengen. Partner eines Serviceautomaten aus einer Klasse $\mathcal{S}$ liegen oft außerhalb von $\mathcal{S}$. $\mathcal{S}_=$ bezeichnet die Menge der Serviceautomaten, die nur Variablen, Konjunktion und die Gleichheitsrelation benutzt. $\mathcal{S}_{=,\neq}$ bezeichnet die Menge der Serviceautomaten, die zusätzlich die Ungleichheitsrelation benutzt.

verschiedenen Werten belegt.

Definition 5.6 (Identitätsäquivalente Belegungen) *Zwei Belegungen β, β' heißen* identitätsäquivalent*, wenn für jedes Paar von Variablen x, y gilt: $\beta(x) = \beta(y)$ gdw. $\beta'(x) = \beta'(y)$.*

Definition 5.7 (Identitätsbewahrendes Prädikat) *Ein Prädikat P heißt* identitätsbewahrend*, wenn jedes Paar $\beta, \beta' \in P$ identitätsäquivalent ist.*

Jedes identitätsbewahrende Prädikat können wir als Konjunktion von Formeln der Form $x = y$ und $x \neq y$ für Variablen x und y schreiben.

Definition 5.8 (Identitätsautomat) *Ein Serviceautomat heißt* Identitätsautomat*, wenn der Guard jeder Kante ein identitätsbewahrendes Prädikat ist.*

Disjunktionen (z. B. $x = y \vee x \neq z$) können nicht durch ein identitätsbewahrendes Prädikat ausgedrückt werden. Alternativen können aber durch explizite Verwendung mehrerer Kanten im Serviceautomaten ausgedrückt werden (z. B. eine Kante mit dem Guard $x = y$ und eine Kante mit dem Guard $x \neq z$).

Ebenso sind Prädikate, welche die Relation zwischen zwei Variablen nicht festlegen, nicht identitätsbewahrend. Das Prädikat $x = x' \wedge y = y'$ legt z. B. die Relation zwischen x und y nicht fest. Es kann aber äquivalent durch die Disjunktion $(x = x' \wedge y = y' \wedge x = y) \vee (x = x' \wedge y = y' \wedge x \neq y)$ dargestellt werden.

Serviceautomaten, deren Prädikate nur aus Variablen, dem Gleichheitszeichen und dem booleschen Operatoren $\wedge, \vee, \neg$ gebildet werden, können wir daher durch Hinzufügen endlich vieler Kanten in einen äquivalenten Identitätsautomaten umwandeln. Wir werden gelegentlich in Darstellungen von Identitätsautomaten die kompaktere Notation

mit booleschen Operatoren verwenden. Für die formalen Beweise ist die äquivalenzklassenbasierte Darstellung der Prädikate in der Regel praktischer.

Man beachte, dass ForceInequal selbst kein Identitätsautomat ist, da im Guard der Kante von start zu choice nicht spezifiziert wird, ob $a_0 = a_1$ oder $a_0 \neq a_1$ gilt. Durch Teilen der Kante in zwei Kanten mit entsprechenden Guards kann der Serviceautomat aber in einen Identitätsautomaten umgewandelt werden.

Identitätsautomaten sind die hinsichtlich der Anzahl der verwendeten Ausdrucksmittel einfachste Klasse von Serviceautomaten mit Variablen, innerhalb der man sinnvolle Aussagen über Partner machen kann.

Das Gleichheitszeichen ist in unserem Formalismus unverzichtbar, da es den Bezug zwischen den Variablen eines Knotens und den Variablen seiner Nachfolgerknoten herstellt. Das Ungleichzeichen ist notwendig, um Ausprägungen von Werten unterscheiden zu können.

Eine Variante von ForceInequal wird in [94] als gefärbtes offenes Petrinetz dargestellt. Die Variante kann in der Petrinetzdarstellung ohne Guards, also insbesondere ohne Verwendung des Gleichheitszeichens formuliert werden. In der Petrinetzdarstellung wird daher besonders deutlich, wie elementar der Serviceautomat ForceInequal ist. Die Petrinetzvariante trifft selbst keine Entscheidung, die vom Inhalt der vom Partner empfangenen Nachrichten abhängt. Die Inhalte der Nachrichten werden durch den Service lediglich weitergeleitet, ohne dass diese vom Service gelesen werden. Ein Partner der Petrinetzvariante muss also Inhalte von Nachrichten unterscheiden, obwohl die Petrinetzvariante selbst Inhalte von Nachrichten nicht beachtet.

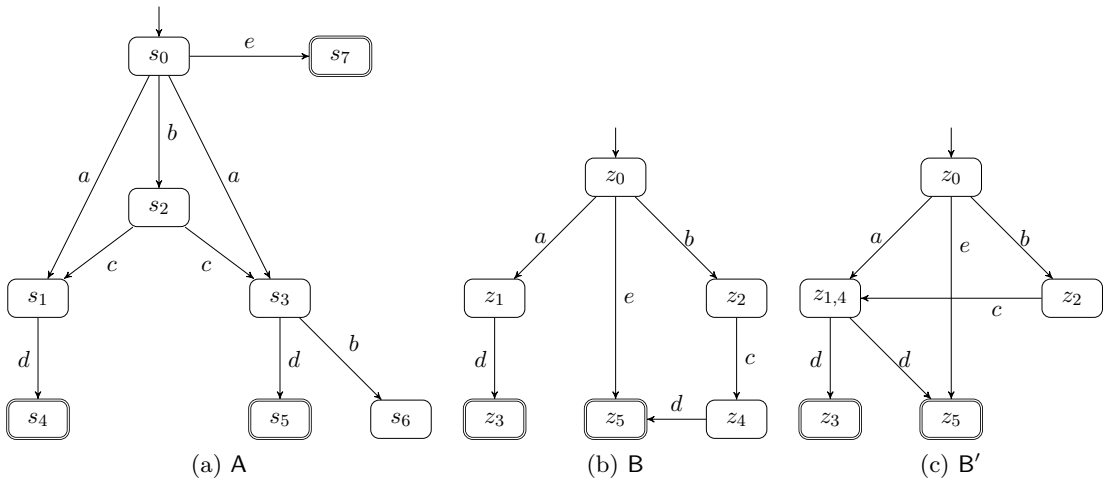
5.3. Eine Transformationsregel für Partner

In diesem Abschnitt stellen wir eine Transformationsregel für Partner vor. Wenn zwei Zustände eines Zustandsautomaten B das gleiche Wissen über einen Partner A von B haben, können diese zu einem Zustand verschmolzen werden. Wir zeigen für azyklische[1] Serviceautomaten, dass die Partnereigenschaft bei dieser Operation erhalten bleibt.

Beispiel Der Zustandsautomat B in Abb. 5.11 ist ein Partner von A. Die Zustände z_1 und z_4 von B haben das gleiche Wissen über A.

$$
\begin{aligned}
&\mathrm{know}_{\mathsf{A},\mathsf{B}}(z_0) = \{s_0\}\\
&\mathrm{know}_{\mathsf{A},\mathsf{B}}(z_1) = \mathrm{know}_{\mathsf{A},\mathsf{B}}(z_4) = \{s_1, s_3\}\\
&\mathrm{know}_{\mathsf{A},\mathsf{B}}(z_2) = \{s_2\}\\
&\mathrm{know}_{\mathsf{A},\mathsf{B}}(z_3) = \{s_4, s_5\}\\
&\mathrm{know}_{\mathsf{A},\mathsf{B}}(z_5) = \{s_4, s_5, s_7\}
\end{aligned}
$$

[1] die Beschränkung auf azyklische Serviceautomaten ist möglicherweise nicht notwendig. Ein allgemeiner Beweis für zyklische Serviceautomaten konnte jedoch nicht gefunden werden.



(a) A (b) B (c) B′

Abbildung 5.11.: Verschmelzen von Zuständen eines Partners. B′ entsteht aus B durch Verschmelzen von z_1 und z_4 zu $z_{1,4}$

Durch Verschmelzen von z_1 und z_4 zu $z_{1,4}$ entsteht der Zustandsautomat B′. Dieser ist auch ein Partner von A. Der Zustandsautomat, der durch Verschmelzen von z_0 und z_4 entsteht, ist dagegen kein Partner von A. In diesem wäre der Zustand s_6 erreichbar.

Wir können am Beispiel feststellen, dass der durch Verschmelzung entstandene Zustand $z_{1,4}$ das gleiche Wissen über A hat wie die beiden Zustände, aus denen er entstanden ist.

$$\text{know}_{\mathsf{A},\mathsf{B}'}(z_{1,4}) = \{s_1, s_3\}$$

Die anderen Zustände haben ebenfalls in B′ das gleiche Wissen wie in B.

Aufgrund dieser Eigenschaft bleibt auch die Eigenschaft von B erhalten, ein Partner von A zu sein.

Wir zeigen zunächst, dass das Wissen der Zustände des Zustandsautomaten durch die Verschmelzung nicht verändert wird. Dies folgt zunächst für die verschmolzenen Zustände aus Korollar 2.6. Analog zu Korollar 2.8 können wir zeigen, dass die Gleichheit des Wissens sich auf den Nachfolgerzuständen fortsetzt.

Im Beweis nutzen wir aus, dass in einem azyklischen Graphen oberhalb der verschmolzenen Knoten keine neuen Pfade entstehen, sondern nur bereits existierende Pfade zusammengefasst werden. Das Verschmelzen eines Knotens mit einem seiner Nachfahren schließen wir dabei explizit aus. Dadurch würde ein Kreis und somit neue, unendlich lange Pfade entstehen.

Lemma 5.4 *Seien A, B interfaceäquivalente azyklische Zustandsautomaten und z_1, z_2 Zustände von B mit* $\text{know}_{A,B}(z_1) = \text{know}_{A,B}(z_2)$*, so dass z_1 kein Nachfahre von z_2 und z_2 kein Nachfahre von z_1 ist. Sei B′ der Zustandsautomat, der aus B durch Verschmelzen*

von z_1 und z_2 zu einem Zustand z_{neu} entsteht. Dann ist

$$\text{know}_{A,B'}(z_{neu}) = \text{know}_{A,B}(z_1) = \text{know}_{A,B}(z_2)$$

und

$$\text{know}_{A,B'}(z) = \text{know}_{A,B}(z)$$

für jeden Zustand $z \neq z_{neu}$ von B'.

Beweis Nach Konstruktion von B' und wegen der Azyklizität von B' entsteht jeder Ablauf zu z_{neu} in B' entweder aus einem Ablauf zu z_1 oder einem Ablauf zu z_2 in B. Daher gilt $\text{Tr}_{B'}(z_{\text{neu}}) = \text{Tr}_B(z_1) \cup \text{Tr}_B(z_2)$.

Daraus folgt mit Korollar 2.6

$$\begin{aligned}\text{know}_{A,B'}(z_{\text{neu}}) &= \text{know}_{A,B}(z_1) \cup \text{know}_A, B(z_2) \\ &= \text{know}_{A,B}(z_1) \\ &= \text{know}_{A,B}(z_2)\end{aligned}$$

Wir zeigen nun $\text{know}_{A,B}(z) = \text{know}_{A,B'}(z)$ für jeden Zustand z von B'.

$\subseteq$: Gilt trivial, da alle in $A \times B$ erreichbaren Zustände auch in $A \times B'$ erreichbar sind.

$\supseteq$: Sei $z_A \in \text{know}_{A,B'}(z)$. Nach Def. (Wissen) ist (z_A, z) in $A \times B'$ erreichbar.
Fall 1: (z_A, z) ist über einen Ablauf von $A \times B'$ erreichbar, der nicht über z_{neu} verläuft. Dann existiert genau dieser Ablauf auch in $A \times B$. Damit ist (z_A, z) in $A \times B$ erreichbar und somit $z_A \in \text{know}_{A,B}(z)$.
Fall 2: (z_A, z) ist nur über einen Ablauf von $A \times B'$ erreichbar, der über z_{neu} verläuft. Dann ist in $A \times B'$ ein Zustand (z'_A, z_{neu}) erreichbar, von dem aus (z_A, z) erreichbar ist. Nach Lemma 2.4 gibt es w so, dass $z'_A \stackrel{w}{\Longrightarrow}_A z_A$ und $z_{\text{neu}} \stackrel{w}{\Longrightarrow}_{B'} z$. Nach Konstruktion von B' ist $z_1 \stackrel{w}{\Longrightarrow}_B z$ oder $z_2 \stackrel{w}{\Longrightarrow}_B z$. Nach Lemma 2.4 ist (z_A, z) von (z'_A, z_1) oder von (z'_A, z_2) aus in $A \times B$ erreichbar.

Nach Vor. ist $\text{know}_{A,B'}(z_{\text{neu}}) = \text{know}_{A,B}(z_1) = \text{know}_{A,B}(z_2)$. Nach Def. (Wissen) folgt damit aus der Erreichbarkeit von (z'_A, z_{neu}), dass auch (z'_A, z_1) und (z'_A, z_2) in $A \times B$ erreichbar sind.

Damit ist (z_A, z) in $A \times B$ erreichbar und somit $z_A \in \text{know}_{A,B}(z)$. □

Dieses Lemma besagt, dass die Menge der erreichbaren Zustände des Kreuzproduktes durch das Verschmelzen der Zustände nicht größer wird. Da von jedem erreichbaren Zustand von $A \times B$ ein Endzustand erreichbar ist, ist dieser erst recht in $A \times B'$ erreichbar.

Theorem 5.5 *Seien A, B interfaceäquivalente azyklische Zustandsautomaten, die Partner sind und z_1, z_2 Zustände von B mit* $\text{know}_{A,B}(z_1) = \text{know}_{A,B}(z_2)$*, so dass z_1 kein Nachfahre von z_2 und z_2 kein Nachfahre von z_1 ist.*

Sei B' der Zustandsautomat, der aus B durch Verschmelzen von z_1 und z_2 zu einem Zustand z_{neu} entsteht. Dann ist B' ein Partner von A.

Beweis Sei (z_A, z'_B) ein in $A \times B'$ erreichbarer Zustand. Wir zeigen: Von (z_A, z'_B) ist ein Endzustand erreichbar. Sei f die Funktion, die z_1 und z_2 auf z_{neu} abbildet und jeden anderen Knoten auf sich selbst. Dann gibt es einen Zustand z_B von B mit $f(z_B) = z'_B$. Aus $z_A \in \text{know}_{A,B'}(z'_B)$ folgt mit Lemma 5.4 $z_A \in \text{know}_{A,B}(z_B)$. Damit ist (z_A, z_B) in $A \times B$ erreichbar.

Nach Voraussetzung ist von (z_A, z_B) in B ein Endzustand (z_A^Ω, z_B^Ω) erreichbar. Nach Lemma 2.4 gibt es ein Wort w mit $z_A \stackrel{w}{\Longrightarrow}_A z_A^\Omega$ und $z_B \stackrel{w}{\Longrightarrow}_B z_B^\Omega$. Da zu jedem Ablauf von B ein Ablauf in B' mit gleichem Trace existiert, gilt auch $f(z_B) \stackrel{w}{\Longrightarrow}_{B'} f(z_B^\Omega)$. Nach Lemma 2.4 ist damit $(z_A^\Omega, f(z_B^\Omega))$ von $(z_A, f(z_B))$ in $A \times B'$ erreichbar. Dieser ist ein Endzustand. □

Man beachte, dass nach Definition der Verschmelzungsoperation z_{neu} ein Endzustand ist, wenn z_1 oder z_2 Endzustände sind.

In Abschnitt 6.1 stellen wir eine Anwendung dieses Theorems vor.

6. Speicherbedarf von Partnern

In diesem Abschnitt untersuchen wir, wie viele Speicherplätze ein Partner eines Serviceautomaten benötigt und wann Speicherplätze weggelassen werden können. Diese Fragestellung ist aus praktischer Sicht interessant, da ein Service sparsam mit seinen Ressourcen umgehen sollte und nicht mehr Speicher verwenden sollte als zur Erfüllung seiner Aufgabe notwendig ist. Aus theoretischer Sicht ist zum anderen eine Untergrenze für die Anzahl der mindestens benötigten Variablen interessant.

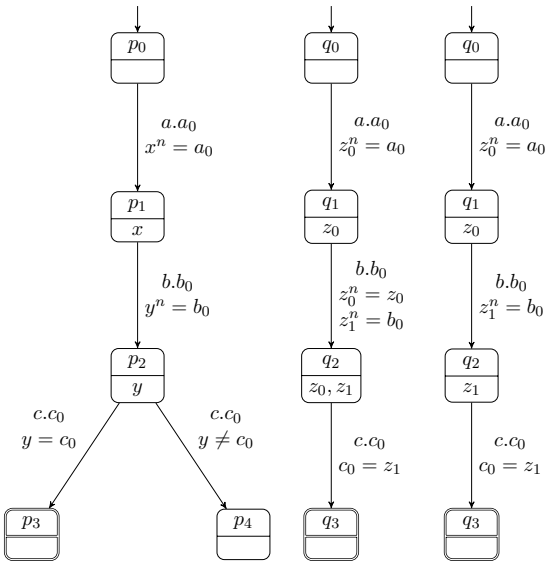
Wir zeigen einerseits, dass in bestimmten Fällen Variablen eines Serviceautomaten überflüssig sind und weggelassen werden können. Wir zeigen andererseits, dass es Serviceautomaten gibt, deren Partner unendlich viele Speicherplätze benötigen. Dies führt zu der wichtigen Erkenntnis, dass es Identitätsautomaten gibt, deren Partner im allgemeinen nicht endlich darstellbar sind.

6.1. Eliminieren redundanter Variablen

In diesem Abschnitt untersuchen wir, wann eine Variable eines Partners eines Serviceautomaten eliminiert werden kann. Eine Variable eines Knotens entspricht einem Speicherplatz des Serviceautomaten. Wir definieren, wann eine Variable eines Serviceautomaten *redundant* ist und zeigen, dass eine redundante Variable eines Partners mit der in Abschnitt 2.4.3 vorgestellten Operation eliminiert werden kann, ohne dass der Partner die Eigenschaft verliert, ein Partner zu sein.

Beispiel Der Serviceautomat A in Abb. 6.1 wählt nichtdeterministisch zwei ganze Zahlen, speichert diese in den Variablen x und y und kommuniziert diese auf a und b. Für welche Aktion A in seinen Endzustand geht, hängt nur von der auf b kommunizierten Zahl ab, nicht aber von der auf a kommunizierten Zahl. Daher ist die Variable z_0, in welcher der Partner B von A die auf a kommunizierte Zahl speichert, intuitiv überflüssig. Die in z_0 gespeicherte Zahl wird nicht dazu benötigt, dass B auf c die Zahl kommunizieren kann, die A in seinen Endzustand bringt. Daher kann z_0 eliminiert werden. Der Serviceautomat B', der aus B hervorgeht, indem z_0 in q_2 eliminiert wird, ist daher ebenfalls ein Partner von A.

Eine Variable x eines Knotens q eines Serviceautomaten B heißt *redundant* bezüglich eines Serviceautomaten A, wenn sie keine Information darüber enthält, in welchem Zustand sich A befindet. Formal bedeutet dies, dass sich das Wissen eines Zustands von B über A nicht ändert, wenn die Belegung von x verändert wird.



(a) A (b) B (c) B′

Abbildung 6.1.: Serviceautomat A und sein Partner B vor und nach Elimination von z_0

Definition 6.1 (Redundante Variable) *Seien A, B interfaceäquivalente Serviceautomaten und q ein Knoten von B. Eine Variable x von q heißt* redundant *(in q bzgl. A), wenn für jedes Paar in B erreichbarer Instanzen (q, β_1) und (q, β_2) von q mit $\beta_1(y) = \beta_2(y)$ für jede Variable y von q mit $x \neq y$ gilt:*

$$\text{know}_{\mathsf{A},\mathsf{B}}(q, \beta_1) = \text{know}_{A,B}(q, \beta_2)$$

Da das Wissen nicht erreichbarer Zustände immer leer ist, fordern wir die Gleichheit des Wissens nur für erreichbare Zustände.

Aus der Definition folgt unmittelbar, dass eine Variable eines Knotens, deren Belegung vollständig durch die Belegung der anderen Variablen des Knotens festgelegt wird, redundant ist. Dann nämlich gibt es zu jeder Belegung dieser Variablen genau eine erreichbare Instanz. Ein Spezialfall eines solchen Zusammenhangs stellen zwei Variablen dar, die stets den gleichen Wert haben. Von diesen ist eine Variable überflüssig. Alpern et al. [3] untersuchen, wie Variablen identifiziert werden können, welche mit dem gleichen Wert belegt werden.

Beispiel Die Variable z_0 ist im Knoten q_2 von B redundant. Für beliebige $i, j \in \mathbb{Z}$ gilt

$$\text{know}_{\mathsf{A},\mathsf{B}}(q_2, z_0 = i, z_1 = j) = \{(p_2, y = j)\}$$

Das Wissen von B über A in q_2 hängt also nicht der Belegung von z_0 ab. z_1 ist in q_2 dagegen nicht redundant. In Knoten q_1 ist z_0 nicht redundant, denn es gilt für jedes

$i \in \mathbb{Z}$

$$\text{know}_{\mathsf{A},\mathsf{B}}(q_1, z_0 = i) = \{(p_2, x = i)\}$$

Der Serviceautomat, der durch Eliminieren einer redundanten Variablen aus einem azyklischen Partner entsteht, ist selbst wieder ein Partner. Daher ist der Serviceautomat B', der aus B durch Eliminieren von z_0 entsteht, auch ein Partner von A.

Im allgemeinen gilt das folgende Theorem:

Theorem 6.1 (Eliminieren einer redundanten Variable) *Seien A, B azyklische Partner, q ein Knoten von B und x eine bezüglich A redundante Variable von q. Kein nicht erreichbarer Zustand von B habe einen Nachfolger.*

Dann ist der Serviceautomat B', der aus B durch Eliminierung von x in q entsteht, ein Partner von A.

Das Eliminieren einer redundanten Variablen entspricht (siehe Korollar 2.10) dem Verschmelzen vieler Knoten mit jeweils gleichem Wissen. Die Korrektheit des Theorems folgt damit direkt aus dem Theorem 5.5 aus Abschnitt 5.3. Da die Serviceautomaten azyklisch sind und je zwei Instanzen eines Knotens nicht Nachfolger füreinander sind, sind die Voraussetzungen zum Anwenden von Theorem 5.5 erfüllt.

Weiterhin ist anzumerken, dass aufgrund der Voraussetzung in Theorem 6.1, dass nicht erreichbare Zustände keine Nachfolger haben, das Verschmelzen eines nicht erreichbaren Zustands effektiv dessen Entfernung entspricht. Diese Feststellung ist wichtig, da ein nicht erreichbarer Zustand immer eine leere Wissensmenge hat und daher nicht die Voraussetzung von Theorem 5.5 erfüllt. Das Verschmelzen eines Zustandes mit einem nicht erreichbaren Zustand, der keinen Nachfolger hat, verändert den Erreichbarkeitgraphen nicht.

In Abschnitt 13.3 zeigen wir, wie für eine Variable eines Knotens eines Identitätsautomaten entschieden werden kann, ob sie redundant ist.

6.2. Partner mit unbeschränkt viel Speicher

Im vorangegangenen Abschnitt haben wir gezeigt, dass Variablen von Knoten eines Partners unter bestimmten Umständen überflüssig sind und eliminiert werden können. In diesem Abschnitt untersuchen wir, wie viele Variablen ein Serviceautomat mindestens benötigt, um ein Partner eines gegebenen anderen Serviceautomaten zu sein.

Diese Fragestellung ist interessant, da ein Service nicht mehr Speicher verwenden sollte, als zur Erfüllung seiner Aufgabe notwendig ist. Wir wollen daher wissen, wie viel Speicherplatz ein Partner eines Serviceautomaten mindestens benötigt.

Definition 6.2 (Speicherbeschränkter Serviceautomat) *Ein Serviceautomat A heißt k-*speicherbeschränkt*, wenn jeder Knoten von A höchstens k Variablen hat.*

Da die Entfaltung eines Serviceautomaten keine Variablen hat, hat jeder bedienbare Serviceautomat immer auch einen Partner ohne Variablen. Schränken wir die Menge der betrachteten Partner auf eine Klasse ein, innerhalb der die Entfaltung nicht darstellbar ist, ist die Frage nach der Anzahl der benötigten Variablen nicht trivial. Die Anzahl der benötigten Variablen hängt offensichtlich von der betrachteten Serviceautomatenklasse ab. Ist es in der Klasse z. B. möglich, durch Gödelisierung einer Sequenz von Werten den Inhalt mehrerer Variablen in einer Variablen zu komprimieren, dann gibt es zu jedem Partner auch einen Partner mit höchstens einer Variablen.

Innerhalb der Klasse der Identitätsautomaten ist Gödelisierung nicht möglich, da die Klasse keine Funktionen erlaubt. Da die einzige Beziehung, die in der Klasse ausgedrückt werden kann, die Gleichheit oder Ungleichheit ist, können zwei Variablen höchstens dann zusammengefasst werden, wenn sie garantiert immer gleich belegt sind.

Jeder Wert, den ein Serviceautomat dieser Klasse selbst erzeugt oder von einem anderen Serviceautomaten erhält, und der nicht gleich einem bereits in einer anderen Variablen gespeicherten Wert ist, muss in einer eigenen Variable gespeichert werden, falls er später noch einmal benutzt werden soll. Es stellt sich die Frage, wie lange ein in einer Variablen gespeicherter Wert gebraucht wird und wann er gelöscht oder überschrieben werden kann.

Intuitiv könnte man annehmen, dass jeder bedienbare Identitätsautomat auch einen Partner in seiner Klasse hat, der in etwa genau so viele Variablen hat wie er selbst. Ein solche Partner könnte den Inhalt jeder Variable des Service spiegeln und beim Kommunizieren von Nachrichten auf diese Bezug nehmen.

Wie der in Abb. 6.2 abgebildete Serviceautomat RandomSeq zeigt, ist dies ein Irrtum. Jeder Partner dieses Serviceautomaten innerhalb der Klasse der Identitätsautomaten hat unendlich viele Knoten und Variablen. Der Serviceautomat RandomSeq selbst hat dagegen nur zwei Variablen:

RandomSeq produziert in den Knoten loop_1 bis loop_4 eine im Wesentlichen zufällige Folge von Wertepaaren, welche über den Kanal a kommuniziert werden. Diese Folge kann beliebig lang werden. Jeder Partner muss die gesamte Folge vollständig speichern.

Durch einen Trick zwingt RandomSeq einen potentiellen Partner dazu, jedes Glied der Folge zu speichern: Unter die Folge wird an einer zufälligen Stelle (Schritt von loop_1 zu loop_2) ein Paar von Werten gemischt, welches zur erfolgreichen Terminierung des Serviceautomaten notwendig ist. Dieses Paar wird in den Variablen k und v gespeichert. Um RandomSeq in seinen Endzustand zu bringen, genügt es, wenn der Partner sich die zweite Komponente dieses einen Paares merkt und später auf Kanal c kommuniziert. Alle anderen Paare sind unwichtig. Für den Partner ist aber zunächst nicht erkennbar, bei welchem Paar der Folge es sich um das relevante handelt. Daher muss der Partner zunächst alle Paare speichern.

RandomSeq kommuniziert im Anschluss an die Folge auf Kanal b eine Information, die es dem Partner erlaubt, das relevante Paar zu identifizieren. Die erste Komponente des Paares dient hierbei als eindeutiger Schlüssel. Kein anderes Paar stimmt mit diesem auf der ersten Komponente überein. Damit ist es dem Partner möglich, aus der gespeicherten Folge das relevante Paar zu finden und genau die Aktion auszuführen, die RandomSeq in seinen Endzustand bringt.

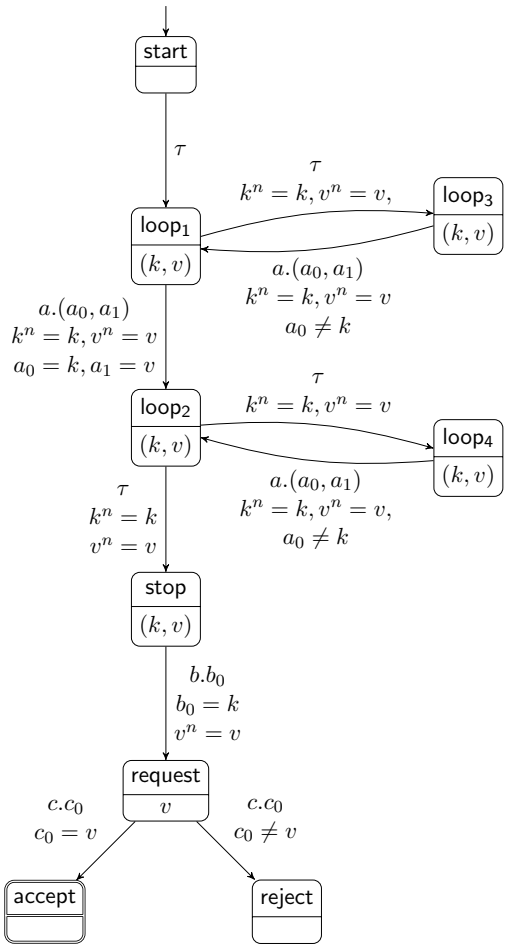


Abbildung 6.2.: Identitätsautomat RandomSeq

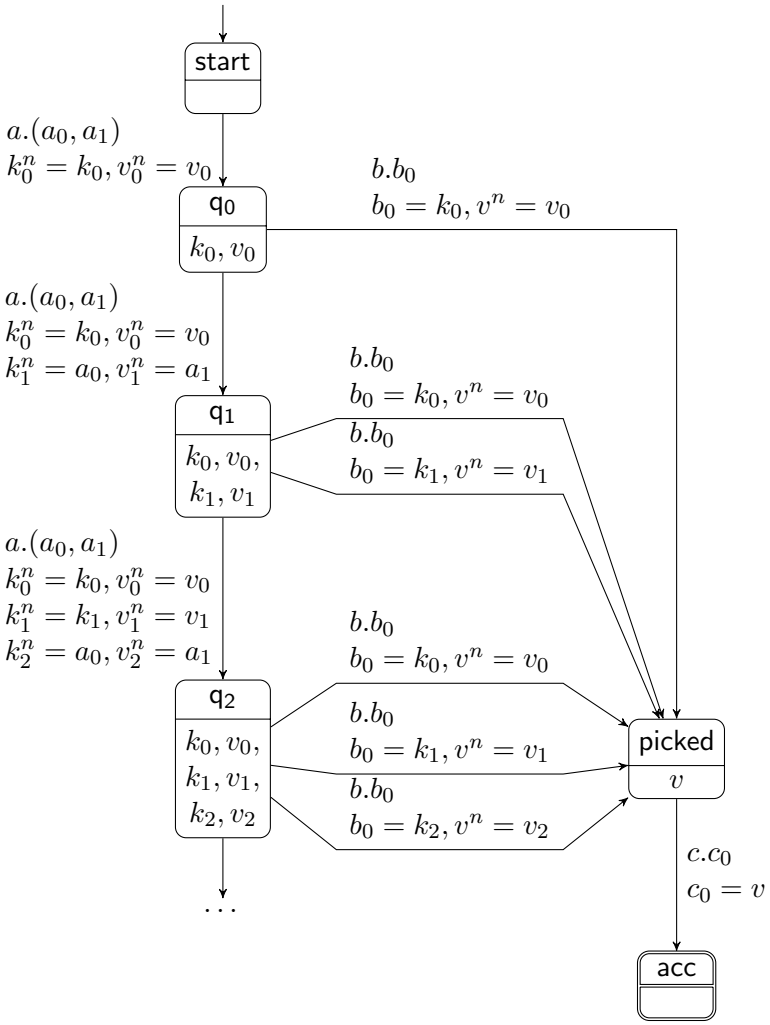


Abbildung 6.3.: Identitätsautomat UnlimitedMemory

Der Serviceautomat UnlimitedMemory ist ein Partner von RandomSeq. UnlimitedMemory speichert jedes auf Kanal a kommunizierte Paar und führt dazu in jedem Schritt zwei neue Variablen ein. UnlimitedMemory hat unendlich viele Knoten und die Anzahl der Variablen pro Knoten wächst unbeschränkt. Zu dem auf b kommunizierten Schlüssel wählt UnlimitedMemory den zugehörigen Wert aus und kommuniziert diesen auf c.

Intuitiv ist klar, dass jeder Partner von RandomSeq sich so ähnlich wie UnlimitedMemory verhält und eine beliebig lange Folge von Paaren speichern muss. Jeder Partner von RandomSeq innerhalb der Klasse der Identitätsautomaten hat daher unendlich viele Variablen.

Wir beweisen nun formal, dass jeder Partner von RandomSeq eine unbeschränkt wachsende Anzahl von Variablen hat. Die Argumentation des Beweises folgt dem Pidgeonhole-Prinzip. Wir nehmen an, dass ein Partner existiert, der höchstens $2n$ Variablen in jedem Knoten hat. Wir betrachten dann eine Folge von $n+1$ Paaren. Mindestens eines der $n+1$ Paare kann der Partner nicht speichern.

Theorem 6.2 *Es gibt einen bedienbaren Identitätsautomaten, so dass kein speicherbeschränkter Identitätsautomat ein Partner von diesem ist.*

Beweis Sei B ein n-speicherbeschränkter Identitätsautomat, der ein Partner von RandomSeq ist. Sei n eine natürliche Zahl. Wir betrachten das Eingabewort $w = a.(0,1)\,a.(2,3)\,a.(4,5)\ldots a.(2n,2n{+}1) \in \Sigma^{n+1}(B)$. Sei (q,β) ein Zustand von B, der in B über die Sequenz w erreichbar ist. q hat nach Voraussetzung höchstens n Variablen. Daher gibt es ein $i \in \{0,\ldots,n\}$, so dass keine der Variablen von q mit dem Wert $2i$ oder $2i+1$ belegt ist.

Andererseits gibt es einen Zustand (p,α) von RandomSeq, der über die Sequenz w erreichbar ist, in dem k mit i und v mit $n+i$ belegt sind (die Belegung von k und v wird nach Konstruktion von RandomSeq nichtdeterministisch gewählt).

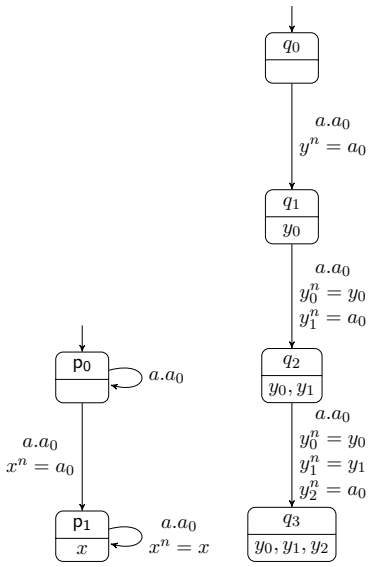
Angenommen, von (p,q,α,β) ist der Komposition ein Endzustand erreichbar. Dann hat B auf dem Weg zum Endzustand zuletzt die Zahl $2i+1$ auf c gesendet. Da aber keine Variable von (q,β) mit $2i+1$ belegt war, muss B die Zahl $2i+1$ irgendwo auf diesem Weg zufällig erzeugt haben. Daher existiert ein ähnlicher [1] Ablauf, in dem statt $2i{+}1$ eine andere Zahl auf c kommuniziert wird. Dieser Ablauf führt dann in den Knoten reject. Dies ist ein Widerspruch zu der Annahme, dass B ein Partner ist. □

Keine der Variablen von UnlimitedMemory ist redundant.

Um besser zu verstehen, warum die gesamte von RandomSeq erzeugte Sequenz gespeichert werden muss, hilft es, ein kleineres Beispiel zu betrachten, welches das von RandomSeq verursachte Problem auf seinen Kern reduziert.

Der Serviceautomat RandomSeq2 in Abb. 6.4 hat nur zwei Knoten und eine einzige Variable x. RandomSeq2 erzeugt wie RandomSeq eine beliebig lange Sequenz zufälliger Werte, die auf dem Kanal a kommuniziert werden. Einer der Werte wird zufällig ausgewählt und in der Variablen x gespeichert. Welcher der Werte ausgewählt wird, wird nicht an die Umgebung kommuniziert.

[1] Diese Ähnlichkeit kann formal durch eine Symmetrie beschrieben werden, vgl. dazu Lemma 13.2



(a) RandomSeq2 (b) NoRedundancy

Abbildung 6.4.: Serviceautomat ohne redundante Variablen

Der Serviceautomat NoRedundancy speichert die ersten drei der auf a kommunizierten Werte in den Variablen y_0, y_1 und y_2. Keine Variable eines Knotens von NoRedundancy ist redundant bzgl. RandomSeq2.
Für den Knoten q_3 und jedes Tripel $i, j, k \in \mathbb{Z}$ gilt beispielsweise:

$$\begin{aligned}&\text{know}_{\mathsf{RandomSeq2}}(q_3, y_0 = i, y_1 = j, y_2 = k) = \\ &\quad \{(p_0), (p_1, x = i), (p_1, x = j), (p_1, x = k)\}\end{aligned}$$

Jede der Variablen y_0, y_1, y_2 enthält mit einer gewissen Wahrscheinlichkeit den Wert, der in x gespeichert ist. Welche der Variablen den gleichen Wert wie x enthält, ist von Ablauf zu Ablauf unterschiedlich. Da keine eindeutige Zuordnung zwischen x und einer der Variablen von NoRedundancy existiert, die mit dem gleichen Wert wie x belegt ist, kann keine der Variablen eliminiert werden, ohne Information über die Belegung von x zu verlieren.

Diese Argumentation gilt in analoger Weise für die Knoten und Variablen des Serviceautomaten UnlimitedMemory.

Der Serviceautomat RandomSeq zeigt, dass ein Partner eines Service im allgemeinen die gesamte Historie der mit dem Service ausgetauschten Nachrichten speichern muss. Für Klassen von Serviceautomaten, in denen es nicht möglich ist, durch Gödelisierung oder ähnliche Techniken Sequenzen von Werten in einer einzigen Variable zu speichern, bedeutet dies, dass Partner im allgemeinen keine endliche Darstellung innerhalb der gleichen Klasse haben.

Für den in Teil 3 vorgestellten Partnersynthesealgorithmus ist es daher notwendig, die Menge der betrachteten Services einzuschränken. Durch die Einschränkung auf azyklische Services können wir gewährleisten, dass ein bedienbarer Service auch einen endlich darstellbaren Partner hat. Die Anzahl der benötigten Variablen ist dann durch die maximale Länge eines Pfades eines azyklischen Serviceautomaten beschränkt.

Das Phänomen, dass einer Variable des Serviceautomaten RandomSeq nicht eindeutig die Variable des Partner zugeordnet werden kann, die den gleichen Wert speichert, ist unabhängig vom Kommunikationsmodell der Serviceautomaten. In welcher Weise die von RandomSeq erzeugten Werte an den Partner kommuniziert werden, spielt für die Beziehungen der Variablen der Serviceautomaten zueinander keine Rolle. In Bezug auf die Bedienbarkeit von RandomSeq ist bei asynchroner Kommunikation allerdings entscheidend, dass alle auf dem Kanal a gesendeten Nachrichten vor dem Empfangen der Nachricht auf Kanal b bereits empfangen wurden. Wird diese Reihenfolge nicht eingehalten, kann der Kanal a von einem Partner wie ein unbeschränkter Speicherplatz verwendet werden, so dass der Partner selbst nur noch endlich viele Variablen braucht.

Durch geeignete Modifikation des Serviceautomaten RandomSeq kann die Einhaltung der Reihenfolge jedoch erzwungen werden. Dazu genügt es, nach dem Senden jeder Nachricht auf a auf eine Empfangsbestätigung des Partners zu warten, aus der erkennbar ist, dass die Nachricht auch tatsächlich vom Partner gelesen wurde. Daher gilt ein zu Theorem 6.2 analoges Theorem auch für asynchron kommunizierende Services.

7. Stand der Technik

In diesem Kapitel geben wir einen Überblick über einige Varianten des Partnerbegriffes sowie mit dem Partnerbegriff verbundene Probleme und Ergebnisse. Die meisten Ergebnisse betreffen Services mit endlich vielen Zuständen und das Kompatibilitätskriterium Deadlockfreiheit.

Kompatibilitätskriterien Neben Deadlockfreiheit und schwacher Terminierung gibt es weitere Kompatibilitätskriterien. *Responsiveness* [93] ist stärker als Deadlockfreiheit und fordert, dass beide Services immer wieder Nachrichten austauschen. Services, die unendlich lange eine interne Schleife ausführen wie der Serviceautomat in Abb. 5.4 werden dadurch ausgeschlossen.

Starke Terminierung [95] fordert, dass ein Service in jedem Ablauf seinen Endzustand garantiert erreicht. Dies ist eine stärkere Forderung als schwache Terminierung, die lediglich fordert, dass von jedem Zustand ein Weg zu einem Endzustand existiert. Schwache Terminierung erlaubt unendlich lange Abläufe, in denen kein Endzustand vorkommt. Weinberg [95] stellt einen Synthesealgorithmus für Partner für starke Terminierung vor, mit dem die Bedienbarkeit eines Service mit endlich vielen Zuständen für starke Terminierung entschieden werden kann.

Mit *temporallogischen Formeln* [72, 49] können zeitliche Abfolgen bestimmter Zustände eines Systems spezifiziert werden. Schwache Terminierung kann durch eine spezielle temporallogische Formel ausgedrückt werden. Die Existenz eines Partners eines Service für ein beliebiges temporallogisch formuliertes Kompatibilitätskriterium zu entscheiden ist ein offenes Problem. Kupferman et al. [48] stellen einen Algorithmus vor, mit dem für spezielle temporallogische Formeln entschieden werden kann, ob ein offenes System in jeder beliebigen Umgebung die spezifizierte Formel erfüllt.

Von Partnern, die ein verhaltensbasiertes Kompatibilitätskriterium erfüllen, kann man zusätzlich fordern, dass sie weitere nicht-funktionale Eigenschaften erfüllen. Jedem Ablauf eines Service kann z. B. eine Ausführungszeit oder Kosten anderer Art zugeordnet werden. Unter allen Partnern des Service möchte man im allgemeinen einen Partner finden, welcher die Kosten der Ausführung minimiert. Dieses Problem wird in [28, 89] behandelt.

Austauschbarkeit Ein Service wird im Laufe seines Lebenszyklus mehrfach aktualisiert und durch neuere Versionen ersetzt. Dieser Austausch wird im allgemeinen unbemerkt von anderen Services, welche mit diesem zusammenarbeiten, vollzogen. Es ist wünschenswert, dass alle diese Services weiterhin mit dem Service zusammenarbeiten können, d. h., die neue Version des Service soll abwärtskompatibel zur alten Version sein. Formal kann Abwärtskompatibilität durch die Austauschbarkeit von Services beschrieben werden. Ein

Service A heißt *austauschbar* durch einen Service A', wenn jeder Partner B der von A auch ein Partner von A' ist.

Van der Aalst [1] und Parnjai [70] stellen einige Transformationsregeln für Services vor, welche die Partnermenge eines Service für das Kompatibilitätskriterium Deadlockfreiheit erhalten.

Die Menge der Partner eines Service ist im allgemeinen unendlich groß. Zu entscheiden, ob ein Service A durch einen anderen Service A' austauschbar ist, ist daher ein nichttriviales Problem. Um Austauschbarkeit zu entscheiden, benötigt man ein Verfahren, um unendlich große Mengen von Partnern zu Vergleichen.

Bedienungsanleitungen (Operating Guidelines) [57] sind eine endliche Repräsentation der Menge aller Partner eines Service für das Kompatibilitätskriterium Deadlockfreiheit. Mit Hilfe von Bedienungsanleitungen kann algorithmisch entschieden werden, ob ein Service ein Partner eines anderen Service ist. Es kann auch algorithmisch entschieden werden, ob die von zwei Bedienungsanleitungen repräsentierten Servicemengen in einer Teilmengenbeziehung stehen. Somit kann mit Bedienungsanleitungen die Austauschbarkeit [86] von Services entschieden werden.

Kaschner [43] stellt eine Repräsentation einer Bedienungsanleitung durch BDDs [13] vor. Mit Hilfe dieser Repräsentation kann für einen Service mit großer Zustandsmenge effizient entschieden werden, ob ein anderer Service ein Partner ist für das Kompatibilitätskriterium Deadlockfreiheit.

Es existieren Varianten der Bedienungsanleitung für Responsiveness [56], starke und schwache Terminierung [95]. Neben Bedienungsanleitungen existieren tracebasierte Ansätze zum Entscheiden der Austauschbarkeit von Services [87, 93].

Weitere Ergebnisse zu Deadlockfreiheit Klai et al. [45] kondensieren ein offenes System, dargestellt durch ein gelabeltes Transitionssystem (LTS), zu einem sogenannten Symbolic Observation Graph (SOG). Der SOG erhält die Information darüber, ob das ursprüngliche LTS deadlockfrei ist. Das Kreuzprodukt zweier SOGs ist isomorph zum SOG des Kreuzproduktes ihrer LTS. Daher können SOGs zum effizienten Testen von Kompositionen offener Systeme auf Deadlockfreiheit verwendet werden.

Findlow [23] stellt eine Transformation für Petrinetze vor, welche die Deadlockfreiheit eines Petrinetzes erhält. Die Arbeit definiert eine Äquivalenzrelation zwischen den Plätzen des Petrinetzes[1]. Verschmelzen äquivalenter Plätze erhält die Eigenschaft des Petrinetzes, deadlockfrei zu sein. Diese Äquivalenzrelation wird lokal durch die Nachbarschaft eines Platzes definiert. Sie kann daher auf die internen Plätzen eines offenen Netzes angewendet werden und ist unabhängig davon, mit welchem anderen offenen Netz dieses komponiert wird. Das Verschmelzen äquivalenter interner Plätze des offenen Netzes erhält die Deadlockfreiheit in jeder Komposition des offenen Netzes mit einem anderen offenen Netz.

[1] Formal ist die Äquivalenz auf den Plätzen der Entfaltung eines gefärbten Petrinetzes definiert, sie kann aber auch direkt auf den Plätzen eines ungefärbten Petrinetzes definiert werden

Ergebnisse zu Daten Die in hier und in Kapitel 5 vorgestellten Kompatibilitätskriterien nehmen keinen expliziten Bezug auf Daten. Daten werden erst zu einem Problem bei der Analyse eines Services.

In [63, 91, 4, 88] werden Diagnoseverfahren zum Auffinden von *Datenflussfehlern* in Workflows vorgestellt. Ein Datenflussfehler liegt beispielsweise vor, wenn auf uninitialisierte Daten lesend zugegriffen wird, d. h., ein Lesezugriff vor dem ersten Schreibzugriff erfolgt. Eine andere Art Datenflussfehler liegt vor, wenn auf einmal geschriebene Daten niemals lesend zugegriffen wird. Bei der Analyse von Datenflussfehlern werden die konkreten Ausprägungen der Daten meist nur in stark abstrahierter Form dargestellt, so dass nicht alle Aspekte des Einflusses von Daten auf das Verhalten des untersuchten Systems im Modell wiedergegeben werden. [91, 4] verwenden Petrinetze zur Analyse der Workflows und berücksichtigen dabei auch die Auswirkungen der Ausprägungen der Daten auf den Kontrollfluss. In [91] kann es durch die Abstraktion von den Ausprägungen der Daten zu fehlerhaften Diagnosen kommen. Um mögliche Fehldiagnosen als solche zu erkennen, verfeinern Heinze et al. [33] den Ansatz von [91] und unterscheiden zwischen *möglichen* und *sicheren* Datenflussfehlern.

Es wäre wünschenswert, entscheiden zu können, ob ein Serviceautomat mit der Nebenbedingung bedienbar ist, dass dieser frei von Datenflussfehlern ist. Die Anwesenheit von Datenflussfehlern wird in der hier zitierten Literatur durch temporallogische Formeln charakterisiert. Mit einem Entscheidungsverfahren für Bedienbarkeit für temporallogische Formeln könnte daher auch die datenflussfehlerfreie Bedienbarkeit entschieden werden.

Die Synthese von Partnern für temporallogische Formeln ist jedoch noch ein offenes Problem. Ein weiteres Problem ist, dass bereits mit den zitierten Verfahren aufgrund der Abstraktion der Daten im allgemeinen nicht entschieden werden kann, ob ein Datenflussfehler vorliegt oder nicht. Um die Bedienbarkeit eines Service zu entscheiden, wären die Verfahren daher nicht genau genug.

8. Zusammenfassung

In diesem Abschnitt haben wir den Begriff des Partners eines Serviceautomaten und den Begriff der Bedienbarkeit eingeführt. Erwartungsgemäß ist die Bedienbarkeit eines Serviceautomaten im allgemeinen unentscheidbar. Die gilt selbst dann, wenn die Erfüllbarkeit jedes Guards des Serviceautomaten entscheidbar ist. Wir untersuchen daher, wie die Ausdrucksfähigkeit von Serviceautomaten geeignet eingeschränkt werden kann, so dass Bedienbarkeit entscheidbar wird.

Wir haben festgestellt, dass Partner eines Serviceautomaten mindestens die Gleichheit bzw. Ungleichheit von Nachrichten unterscheiden können müssen. Dies motiviert die Definition von *Identitätsautomaten*, einer Klasse von Serviceautomaten, die nur Variablen vergleichen kann und keine Funktionen, Konstanten oder Prädikate außer der Gleichheitsrelation und deren Negation benutzt. Es zeigt sich leider, dass es bereits in dieser einfachen Klasse Serviceautomaten gibt, die bedienbar sind, deren Partner jedoch nicht endlich darstellbar sind. Diese Beobachtung betrifft jedoch nur zyklische Serviceautomaten. Für azyklische Serviceautomaten werden wir im folgenden Teil 3 Algorithmen vorstellen, die zu jedem bedienbaren Serviceautomaten einen endlich darstellbaren Partner synthetisieren.

Wir haben außerdem zwei Transformationsregeln für Partner vorgestellt. Wir haben gezeigt, dass unter bestimmten Bedingungen Zustände von Services miteinander verschmolzen werden können und Variablen eliminiert werden können. Diese Transformationen helfen, die Größe bzw. Komplexität eines Service zu reduzieren. Wir haben offen gelassen, wie die Voraussetzungen zum Anwenden der Transformationsregeln auf Services mit unendlich vielen Zuständen überprüft werden können. In Abschnitt 13.3 werden wir zeigen, wie die Transformationsregel zum Verschmelzen von Knoten speziell auf Identitätsautomaten angewendet werden kann.

Teil III.

Synthese von Partnern

9. Motivation und Anwendungen

Wir haben im vorangegangenen Teil gesehen, dass die Existenz von Partnern im allgemeinen unentscheidbar ist. Für Klassen von Serviceautomaten, für die Bedienbarkeit entscheidbar ist, wollen wir die Bedienbarkeit algorithmisch entscheiden können. Die Existenz eines Partners zeigen wir konstruktiv durch die *Synthese* eines solchen.

Für die Partnersynthese gibt es wichtige Anwendungen. Zum einen dient sie zur Plausibilitätsprüfung während der Entwicklung eines Service. Die Bedienbarkeit eines Services ist eine elementare Eigenschaft eines Services, die in Isolation überprüft werden kann, ohne Annahmen über das Verhalten der Umgebung zu machen.

Ein Service, der nicht bedienbar ist, ist inhärent fehlerhaft. Ein nicht bedienbarer Service zeigt in jeder beliebigen Umgebung, in der er ausgeführt wird, fehlerhaftes Verhalten. Das Überprüfen der Bedienbarkeit erlaubt es, Fehler im Entwurf eines Service zu entdecken, die eine korrekte Interaktion mit dem Service unmöglich machen.

Die Bedienbarkeit eines Service allein ist noch nicht ausreichend, um zu garantieren, dass der Service in allen Umgebungen das gewünschte Verhalten zeigt. Es existieren genauere Verfahren, welche die Diagnose von Fehlern im Interaktionsverhalten ermöglichen [54]. Es ist weiterhin möglich, einen fehlerhaften Service automatisch so zu korrigieren, dass er mit einem anderen Service zusammenarbeiten kann [53].

Eine weitere wichtige Anwendung ist die Synthese von *Adaptern*. Viele Services können allein deshalb nicht miteinander zusammenarbeiten, weil ihre Interfaces nicht zueinander kompatibel sind. In diesem Fall kann ein Adapter dazu verwendet werden, um die Interfaces miteinander zu verbinden. Ein solcher Adapter kann nicht nur inkompatible Interfaces überbrücken, sondern auch Verhaltenskompatibilität herstellen. Ein verhaltensbasierter Adapter passt die Kommunikation der verbundenen Service so an, dass ein Kompatibilitätskriterium erfüllt wird.

Im Kern ist ein solcher Adapter ein Partner der beiden Services, die er verbindet, d. h., die Komposition der beiden Services und des Adapters erfüllt das Kompatibilitätskriterium. Ein Adapter kann daher mit Hilfe eines Algorithmus zur Partnersynthese konstruiert werden [27]. Häufig ist es wünschenswert, dass der synthetisierte Adapter zusätzliche Bedingungen bezüglich des erlaubten Verhaltens oder der entstehenden Kosten [26] einhält.

In Kapitel 12 zeigen wir exemplarisch einen Adapter, der eine Inkompatibilität im Format der ausgetauschten Daten zwischen zwei Services beseitigt. Auf die technischen Aspekte der Synthese eines solchen Adapters gehen wir dabei nur oberflächlich ein.

Partnersynthese ist ein Spezialfall der *Controllersynthese*. Die *Kontrolltheorie* befasst sich mit der Steuerung von Fertigungsanlagen (plants). Ein *Regler* (controller) wird dafür verwendet, die Fertigungsanlage zu steuern. Der Regler erhält Informationen von Sensoren, welche die Fertigungsanlage überwachen. Auf Grundlage dieser Informatio-

nen steuert der Regler das Verhalten der Fertigungsanlage. Im allgemeinen sind nicht alle Aspekte des Verhaltens der Anlage steuerbar. Die zahlreichen in der Controllersynthese verwendeten Modelle und Ansätze unterscheiden sich darin, welche Aktionen der Fertigungsanlage gesteuert werden können und welche Ereignisse oder anderweitige Informationen über den Zustand der Fertigungsanlage beobachtbar sind.

Ein Partner kann als spezieller Regler betrachtet werden, der das Verhalten eines anderen Service steuert.

In diesem Teil der Arbeit zeigen wir die Synthese eines Partners für zwei Klassen von Services mit unendlich vielen Zuständen. Wir definieren die Klasse der Serviceautomaten, deren Prädikate mit Formeln der Prädikatenlogik erster Stufe notiert werden. Diese Serviceautomaten nennen wir *Formelautomaten*. Wir zeigen, dass für spezielle azyklische Formelautomaten, deren Formeln in einer entscheidbaren Theorie wie z. B. Presburger-Arithmetik formuliert sind, Bedienbarkeit entschieden werden kann.

Wir präsentieren in Kapitel 13 einen weiteren Synthesealgorithmus für azyklische Identitätsautomaten. Identitätsautomaten sind ein Spezialfall von Formelautomaten. Für Identitätsautomaten können wir die Transformationsregel (Abschnitt 5.5) zum Verschmelzen von Zuständen eines Partners auf Knoten des Partners verallgemeinern. Durch Verschmelzen von Knoten könnte es unter bestimmten Umständen möglich sein, aus einem baumförmigen Partner einen zyklischen Partner zu erzeugen. Die Forschung ist hierzu jedoch nicht abgeschlossen.

10. Algorithmische Grundlagen der Partnersynthese

In diesem Kapitel stellen wir einen Synthesealgorithmus für Partner eines azyklischen Serviceautomaten vor. Wir beschränken uns dabei zunächst auf Zustandsautomaten mit endlich vielen Zuständen. In den Kapiteln 11 und 13 verallgemeinern wir später den Algorithmus auf Serviceautomaten mit unendlich vielen Zuständen.

Der in diesem Kapitel vorgestellte Algorithmus ist eine Variante des von Wolf in [96] vorgestellten Algorithmus. Der ursprüngliche Algorithmus wurde für asynchron kommunizierende endliche Automaten formuliert, welche auch Zyklen enthalten dürfen. Der Algorithmus von Wolf konstruiert zunächst einen *Überapproximation* genannten endlichen Automaten, aus dem durch Entfernen von Zuständen ein Partner erzeugt wird.

Eine endlich darstellbare Verallgemeinerung dieser Überapproximation für Serviceautomaten mit endlich vielen Zuständen konnte in dieser Arbeit nicht gefunden werden. Unser Algorithmus benutzt daher eine Variante der Überapproximation, die speziell auf azyklische Serviceautomaten ausgelegt ist.

Die Korrektheit der in den späteren Abschnitten vorgestellten Algorithmen kann auf die Korrektheit dieses Algorithmus zurückgeführt werden.

Die Arbeitsweise des Algorithmus illustrieren wir anhand des Zustandsautomaten A in Abb. 10.1.

Zu einem gegebenen Zustandsautomaten A wird zunächst ein Serviceautomat U_0 konstruiert, welchen wir als *Überapproximation* der Partnermenge bezeichnen. Aus diesem werden solange Zustände entfernt, bis entweder alle Zustände entfernt wurden (genau dann ist der Serviceautomat nicht bedienbar) oder der resultierende Serviceautomat ein

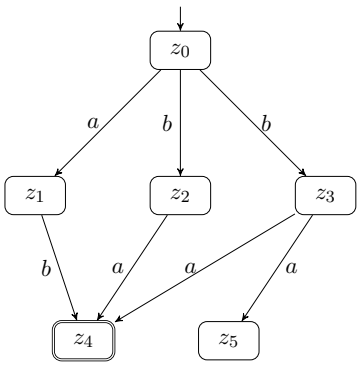
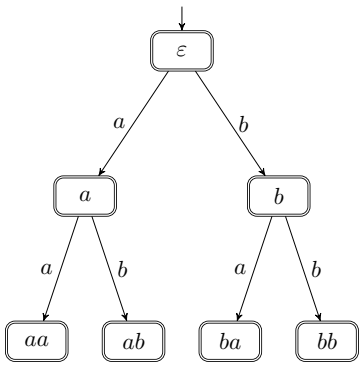


Abbildung 10.1.: Ein Zustandsautomat A



z	$\text{know}_{\mathsf{A},\mathsf{U}_0}(z)$
ε	$\{z_0\}$
a	$\{z_1\}$
b	$\{z_2, z_3\}$
aa	$\emptyset$
ab	$\{z_4\}$
ba	$\{z_4, z_5\}$
bb	$\emptyset$

Abbildung 10.2.: Kanonische Überapproximation U_0

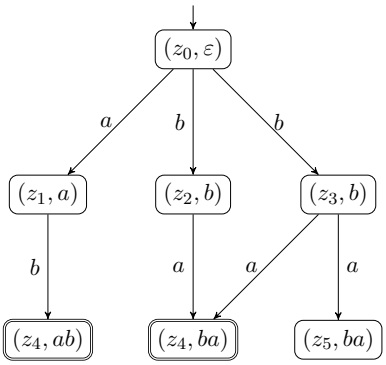


Abbildung 10.3.: Kreuzprodukt $\mathsf{A} \times \mathsf{U}_0$

Partner ist (genau dann ist der Service bedienbar).

Der ursprüngliche Algorithmus von Wolf konstruiert die Überapproximation aus Mengen von Zuständen des gegebenen Automaten. In unserer Variante des Algorithmus ist die Überapproximation ein Baum mit beschränkter Tiefe. Die Tiefe des Baumes richtet sich nach der maximalen Länge eines Ablaufs des gegebenen Zustandsautomaten A.

Definition 10.1 (Kanonische Überapproximation der Partnermenge) *Sei A ein azyklischer Serviceautomat der Tiefe k. Dann nennen wir den kanonischen Baum U_0 zu A mit der Sprache $L = \Sigma^k(A)$ die* kanonische Überapproximation *der Partnermenge von A.*

Abb. 10.2 zeigt die kanonische Überapproximation U_0 der Partnermenge von A aus Abb. 10.1.

Aus der Überapproximation entfernen wir iterativ *schlechte* Zustände. Einen Zustand von U nennen wir *schlecht*, wenn er in einem Zustand des Kreuzproduktes vorkommt, von dem kein Endzustand erreichbar ist.

Definition 10.2 (Guter/schlechter Zustand) *Seien A, B interfaceäquivalente Serviceautomaten. Ein Zustand z_B von B heißt* gut *(bzgl. A), wenn für jeden Zustand $z_A \in \text{know}_{A,B}(z_B)$ gilt: Von (z_A, z_B) ist ein Endzustand in $A \times B$ erreichbar. Andernfalls nennen wir den Zustand* schlecht.

Beispiel Der Zustand *ba* von $\mathsf{U_0}$ ist ein schlechter Zustand, da von (z_5, ba) kein Endzustand erreichbar ist. Alle anderen Zustände von $\mathsf{U_0}$ sind gut.

Ein Partner hat nach Definition nur gute Zustände.

Korollar 10.1 *Zwei Serviceautomaten A und B sind Partner gdw. jeder Zustand von B gut bzgl. A ist.*

Das Syntheseverfahren besteht darin, solange schlechte Zustände aus der Überapproximation zu entfernen, bis nur noch gute Zustände übrig sind oder alle Zustände entfernt wurden. Den erzeugten Partner können wir als Fixpunkt einer Folge kleiner werdender Serviceautomaten beschreiben. Den Fixpunkt nennen wir den *kanonischen Partner*.

Definition 10.3 (Kanonischer Partner) *Sei A ein azyklischer Zustandsautomat der Tiefe k. Sei $U_0, U_1, U_2, \ldots$ eine Folge von Serviceautomaten so, dass U_0 die kanonische Überapproximation der Partnermenge von A ist und U_{i+1} entstehe aus U_i durch Entfernen aller bzgl. A schlechten Zustände von U_i. Sei U^* der Fixpunkt der Folge, d. h. dasjenige U_i mit dem kleinsten Index i, das keinen schlechten Zustand enthält. Dann nennen wir U^* den* kanonischen Partner *von A.*

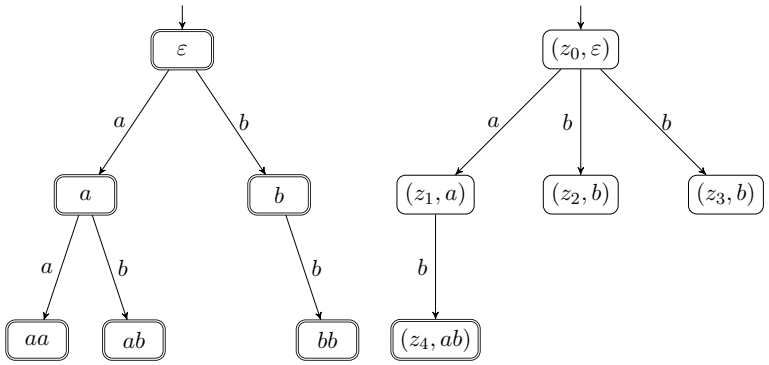
Die Synthese eines kanonischen Partners beschreiben wir explizit in Algorithmus 1.

Algorithmus 1 Synthese eines kanonischen Partners

Eingabe: Ein azyklischer Zustandsautomat A mit Tiefe $k \in \mathbb{N}$
U_0 := kanonische Überapproximation der Partnermenge von A
z_0 := Anfangszustand von U_0
$i := 0$
while es ex. ein schlechter Zustand $z \in Z(U_i)$:
    U_{i+1} := Einschränkung von U_i auf die guten Zustände von U_i
    $i := i + 1$
if $z_0 \in Z(U_i)$
    return U_i
else
    return „A ist nicht bedienbar“

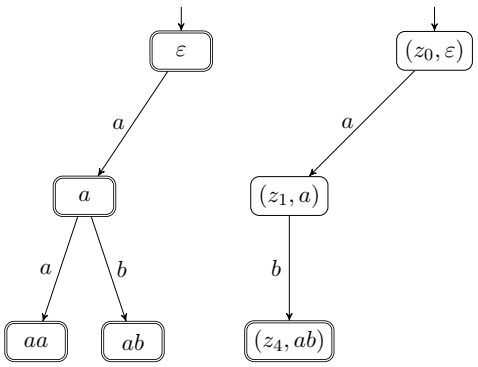
Beispiel Durch Entfernen von *bb* entsteht aus $\mathsf{U_0}$ der Serviceautomat $\mathsf{U_1}$ in Abb. 10.4. In $\mathsf{U_1}$ ist b ein schlechter Zustand, da von (z_2, b) und (z_4, b) kein Endzustand erreichbar ist. Durch Entfernen von b (einschließlich seines Nachfolgers *bb*) entsteht der Serviceautomat $\mathsf{U_2}$ in Abb. 10.5. $\mathsf{U_2}$ ist ein Partner von A.

Der Algorithmus konstruiert genau dann einen Partner, wenn ein solcher existiert. Dies zeigen wir im nächsten Abschnitt.



(a) U_1 (b) $\mathsf{A} \times \mathsf{U}_1$

Abbildung 10.4.: Erster Iterationsschritt der Partnersynthese



(a) U_2 (b) $\mathsf{A} \times \mathsf{U}_2$

Abbildung 10.5.: Zweiter (und letzter) Iterationsschritt der Partnersynthese

10.1. Korrektheit des Algorithmus

Wir zeigen nun die Korrektheit des Algorithmus. Der Beweis folgt der Argumentation in [96].

Wir definieren zunächst die *Matchingrelation* zwischen zwei Serviceautomaten. Die Matchingrelation setzt Zustände mit gleichen Traces in Relation. Die hier vorgestellte Definition unterscheidet sich von der Definition in [96, 57] dadurch, dass τ-Schritte anders als kommunizierende Schritte behandelt werden. Diese Anpassung ist notwendig, da unsere Überapproximation im Gegensatz zu der dort vorgestellten keine τ-Schleifen enthält.

Definition 10.4 (Matchingrelation) *Seien A und B interfaceäquivalente Serviceautomaten. Die Anfangszustände der Serviceautomaten seien z_0^A bzw. z_0^B. Dann nennen wir*

$$\varrho \underset{Def}{=} \{(z_A, z_B) \in Z(A) \times Z(B) |\ \textit{es ex. } w \textit{ mit } z_0^A \stackrel{w}{\Longrightarrow}_A z_A, z_0^B \stackrel{w}{\Longrightarrow}_B z_B\}$$

die Matchingrelation *zwischen A und B.*

Wir fixieren im folgenden einen azyklischen Zustandsautomaten A der Tiefe k. Jeder Partner von A tauscht mit A höchstens k Nachrichten aus, da A seinen Endzustand nach höchstens k Schritten erreicht. Um Bedienbarkeit zu entscheiden, ist es daher ausreichend, nur Serviceautomaten zu betrachten, deren Traces höchstens die Länge k haben. Ein Partner von A kann Abläufe mit längeren Traces haben, diese Abläufe werden aber im Kreuzprodukt mit A nie ausgeführt.

Lemma 10.2 *Sei A ein azyklischer Serviceautomat der Tiefe k. Dann ist A bedienbar gdw. es einen Partner P von A gibt mit* $\mathrm{Tr}(P) \subseteq \Sigma^k(A)$.

Die kanonische Überapproximation U_0 simuliert jeden Partner P von A, der höchstens k kommunizierende Schritte macht und die Matchingrelation ist eine schwache Simulationsrelation.

Lemma 10.3 *Sei P ein azyklischer Partner von A mit* $\mathrm{Tr}(P) \subseteq \Sigma^k(A)$ *und sei ϱ die Matchingrelation zwischen P und U_0. Dann ist ϱ eine schwache Simulationsrelation.*

Da jeder Ablauf der Überapproximation genau einem Trace entspricht, können wir Lemma 2.6 mit Hilfe der Matchingrelation wie folgt umformulieren.

Lemma 10.4 *Sei P ein zu A interfaceäquivalenter Serviceautomat und ϱ die Matchingrelation zwischen P und U_0. Dann gilt*

$$\mathrm{know}_{A,P}(z_P) = \bigcup_{(z_P, z_U) \in \varrho} \mathrm{know}_{A,U_0}(z_U)$$

Dieses Lemma entspricht Lemma 4 aus [57].

Zustände der Überapproximation, die mit einem Zustand eines beliebigen Partners P von A in Relation stehen, werden vom Algorithmus nicht entfernt.

Lemma 10.5 *Sei P ein schwach terminierender Partner von A. Sei U_i ein Serviceautomat der in Def. 10.3 definierten Folge. Sei ϱ die Matchingrelation zwischen P und U_i und $(z_P, z_U) \in \varrho$. Dann ist z_U ein guter Zustand von U_i bzgl. A.*

Beweis Sei $z_A \in \text{know}_{A,U_i}(z_U)$. Aus Lemma 10.4 folgt $z_A \in \text{know}_{A,P}(z_P)$. Da A und P Partner sind, ist von (z_A, z_P) ein Endzustand (z_A^F, z_P^F) in $A \times P$ erreichbar. Nach Lemma 2.4 gibt es ein Wort w mit $z_A \stackrel{w}{\Longrightarrow}_A z_A^F$ und $z_P \stackrel{w}{\Longrightarrow}_P z_P^F$. Wegen schwacher Simulation gibt es ein Paar $(z_P^F, z_U^F) \in \varrho$ mit $z_U \stackrel{w}{\Longrightarrow}_U z_U^F$. Nach Lemma 2.4 ist (z_A^F, z_U^F) in $A \times U$ von (z_A, z_U) erreichbar. Nach Def. (guter Zustand) ist damit z_U ein guter Zustand von U_i. □

Falls A mindestens einen Partner hat, wird insbesondere der Anfangszustand von A nicht entfernt, da dieser mit dem Anfangszustand des Partners in Relation steht. Andererseits ist jeder Fixpunkt der Folge, der mindestens einen Zustand hat, ein Partner. Dies folgt direkt aus der Definition des guten Zustandes.

Daraus folgt unmittelbar die Korrektheit des Algorithmus.

Theorem 10.6 *Sei z_0 der Anfangszustand von A. Dann ist $z_0 \in Z(U^*)$ gdw. A ist bedienbar.*

Der kanonische Partner ist sogar ein permissivster Partner von A unter allen Partnern P mit $\text{Tr}(P) \subseteq \Sigma(A)^k$. Gemäß Lemma 10.5 wird im speziellen kein Zustand der Überapproximation entfernt, der mit einem Zustand eines permissivsten Partners in Relation steht. Da die Matchingrelation eine schwache Simulationsrelation ist, simuliert der kanonische Partner den permissivsten Partner. Daher ist dieser selbst ein permissivster Partner.

10.2. Synthese für asynchron kommunizierende Serviceautomaten

Der vorgestellte Synthesealgorithmus kann in leicht modifizierter Form dafür verwendet werden, einen Partner eines asynchron kommunizierenden azyklischen Serviceautomaten A zu synthetisieren. Dafür werden die Sende- und Empfangskanäle von A mit ihrem jeweiligen Nachrichtenpuffer (siehe Abschnitt 2.3.1) synchron komponiert. Die Komposition aller zu A gehörigen Nachrichtenpuffer notieren wir mit $\text{MsgBuf}(A)$. Um einen einen Serviceautomaten B zu synthetisieren, so dass $A \otimes \text{MsgBuf}(A) \otimes B$ schwach terminiert, genügt es, den Synthesealgorithmus auf die Eingabe $A \otimes \text{MsgBuf}(A)$ anzuwenden.

Dieses Vorgehen bedarf einer kurzen Rechtfertigung, da $A \otimes \text{MsgBuf}(A)$ nicht azyklisch ist und im allgemeinen unendlich viele Zustände hat. Die Anzahl der Nachrichten, die A mit einem beliebigen Partner austauschen kann, ist genau wie im synchronen Fall

beschränkt durch die Tiefe k von A. Jeder Nachrichtenpuffer muss im Endzustand leer sein. Um die Bedienbarkeit von A zu entscheiden, genügt es daher, die Existenz eines Partners zu zeigen, der maximal k Nachrichten sendet oder empfängt.

Lemma 10.7 *Sei A ein azyklischer asynchron kommunizierender Serviceautomat mit der Tiefe k, und* $\mathrm{MsgBuf}(A)$ *die synchrone Komposition der Nachrichtenpuffer von A. Dann hat $A \otimes \mathrm{MsgBuf}(A)$ einen Partner (nach Def. 5.3) gdw. es einen azyklischen Serviceautomaten B der Tiefe k gibt, so dass $A \otimes \mathrm{MsgBuf}(A) \otimes B$ schwach terminiert.*

Daher kann genau wie im synchronen Fall der kanonische Baum der Tiefe k als Überapproximation U_0 der Partnermenge verwendet werden. Dieser simuliert jeden Serviceautomaten B, so dass B höchstens k Nachrichten kommuniziert und $A \otimes \mathrm{MsgBuf}(A) \otimes B$ schwach terminiert. Die Komposition $A \otimes \mathrm{MsgBuf}(A) \otimes U_0$ ist azyklisch. Wenn A ein endlich großes Alphabet und endlich viele Zustände hat, gilt dies auch für die Komposition $A \otimes \mathrm{MsgBuf}(A) \otimes U_0$. Daher sind alle Zustandsräume, die der Algorithmus berechnet, endlich groß. Daher kann ein mit A asynchron kommunizierender Partner B effektiv berechnet werden.

11. Partnersynthese für Formelautomaten

Im vorangegangenen Kapitel haben wir die Grundlagen der Partnersynthese anhand von Zustandsautomaten mit endlich vielen Zuständen kennen gelernt. Wir wollen uns nun Servieautomaten mit unendlich vielen Zuständen zuwenden. Wie in Abschnitt 5.1 gezeigt wurde, ist Bedienbarkeit für Serviceautomaten im Allgemeinfall unentscheidbar. Daher können wir das Syntheseproblem nur dann lösen, wenn wir die Menge der betrachteten Serviceautomaten einschränken.

In diesem Kapitel lösen wir das Syntheseproblem für eine Klasse azyklischer Serviceautomaten, deren Prädikate in der Prädikatenlogik erster Stufe darstellbar sind. Diese Serviceautomaten nennen wir *Formelautomaten*.

Die Prädikatenlogik der ersten Stufe ist im allgemeinen unentscheidbar. Es gibt jedoch entscheidbare Spezialfälle (siehe [67] für einen Überblick). Zu diesen gehören z. B. die Theorien der Prädikatenlogik erster Stufe der booleschen Algebren, der reell abgeschlossenen Körper, der algebraisch abgeschlossenen Körper und der Abelschen Gruppen. Von besonderem Interesse ist außerdem *Presburger-Arithmetik*, da sie vergleichsweise ausdrucksstark ist.

In diesem Kapitel stellen wir einen Synthesealgorithmus vor, der unendlich große Mengen von Zuständen mit prädikatenlogischen Formeln beschreibt. Um ausdrücken zu können, dass unendlich viele Zustände alle bei der Synthese relevanten Eigenschaften erfüllen, sind Quantoren zwingend erforderlich.

Im folgenden Abschnitt 11.1 definieren wir die Grundlagen der Prädikatenlogik erster Stufe. In Abschnitt 11.3 formulieren wir grundlegende Eigenschaften eines Serviceautomaten mit Hilfe prädikatenlogischer Formeln. Den Algorithmus zur Synthese eines Partners stellen wir in Abschnitt 11.4 vor.

11.1. Prädikatenlogische Formeln

In Abschnitt 2.1 haben wir ein Prädikat als Menge von Belegungen definiert. Dabei haben wir zunächst bewusst offen gelassen, in welchem Formalismus die Prädikate dargestellt werden. Je nach betrachteter Automatenklasse und Fragestellung sind unterschiedliche Darstellungen sinnvoll. Die konkrete Darstellung können wir jeweils passend zur verwendeten Analysemethode oder Berechnungsmethode wählen.

In diesem Abschnitt definieren wir zunächst allgemein die *prädikatenlogischen Formeln* der Prädikatenlogik erster Stufe. In der Prädikatenlogik erster Stufe können wir die *Presburger-Arithmetik* formulieren. Presburger-Arithmetik ist eine entscheidbare Theorie. Für diesen Spezialfall ist es algorithmisch möglich, die prädikatenlogischen Formeln auf Erfüllbarkeit zu prüfen. Der mit der Prädikatenlogik erster Stufe vertraute Leser kann diesen Abschnitt überspringen und bei Bedarf zu einem späteren Zeitpunkt lesen.

Die Terme und Formeln der Prädikatenlogik erster Stufe werden über der Signatur gebildet, die aus den folgenden Zeichen besteht:

- den Quantoren $\forall$ und $\exists$,
- den logischen Operatoren $\wedge, \vee, \implies, \iff$ und $\neg$,
- den logischen Konstanten TRUE und FALSE
- den Klammersymbolen (und)
- dem Gleichheitszeichen $=$,
- einer Menge von Variablen $\mathcal{X} = \{x_0, x_1, \dots\}$,
- einer Menge von Funktionssymbolen $\mathcal{F} = \{F_0, F_1, \dots\}$.
- einer Menge von Relationssymbolen $\mathcal{R} = \{R_0, R_1, \dots\}$.

Jedes Funktionssymbol F_i und jedes Relationssymbol R_i hat eine Stelligkeit $n \in \mathbb{N}$. Funktionssymbole der Stelligkeit 0 nennen wir *Konstanten*.

Über dieser Signatur bilden wir die Menge der Terme. Im Rahmen dieser Arbeit beschränken wir uns auf eine Prädikatenlogik mit nur einer Sorte, d. h. alle Terme haben den gleichen Typ. Dies schränkt die Möglichkeiten bei der Modellierung von Serviceautomaten stark ein, ist aber für die in der Arbeit behandelten Probleme ausreichend.

Definition 11.1 (Term) *Die Menge* TERM *der* Terme *ist induktiv definiert.*

- *Sei $x \in \mathcal{X}$ eine Variable. Dann ist x ein Term.*
- *Sei F ein n-stelliges Funktionssymbol, $T_1, \dots, T_n$ Terme. Dann ist $F(T_1, \dots T_n)$ ein Term.*

Aus den Termen bilden wir *Formeln*. Wie üblich bezeichnen wir eine Variable in einem Term als *frei*, wenn sie im Geltungsbereich keines Quantors vorkommt.

Definition 11.2 (Formel) *Die Menge* FORM *der Formeln ist wie folgt definiert.*

- TRUE *und* FALSE *sind Formeln.*
- *Seien $T_1, T_2 \in$* TERM*. Dann ist $T_1 = T_2$ eine Formel.*
- *Seien $T_1, \dots, T_n \in$* TERM*, R ein n-stelliges Relationssymbol. Dann ist $R(T_1, \dots, T_n)$ eine Formel.*
- *Seien T_1, T_2 Formeln. Dann sind $T_1 \wedge T_2, T_1 \vee T_2, T_1 \implies T_2$ und $T_1 \iff T_2$ Formeln.*
- *Sei T eine Formel. Dann ist $\neg T$ eine Formel.*

- *Sei T eine Formel, x eine Variable, die frei in T vorkommt. Dann sind $\forall x : T$ und $\exists x : T$ Formeln.*

Die Bedeutung der Funktionssymbole in $\mathcal{F}$ und Relationssymbole in $\mathcal{R}$ ist abhängig von ihrer *Interpretation*. Die booleschen Verknüpfungen werden stets mit der üblichen Semantik interpretiert.

Definition 11.3 (Interpretation) *Eine* Interpretation *I ist ein Zuordnung, die eine Menge $\mathcal{U}$, das* Universum *fixiert und*

- *jedem n-stelligen Funktionssymbol $F \in \mathcal{F}$ eine n-stellige Funktion $F^I : \mathcal{U}^n \to \mathcal{U}$ und*
- *jedem n-stelligen Relationssymbol R eine n-stellige Relation $R^I \subseteq \mathcal{U}^n$*

zuordnet.

Die Menge aller Funktionen F^I und die Menge aller Relationen R^I bildet zusammen mit der Trägermenge $\mathcal{U}$ eine *Struktur*.

Für eine gegebene Interpretation evaluiert ein Term für eine Belegung zu einem Element in $\mathcal{U}$ und eine Formel zu einem der Wahrheitswerte true oder false.

Definition 11.4 (Evaluierungsfunktion) *Seien I eine Interpretation und β eine Belegung. Die* Evaluierungsfunktion $\mathrm{eval}_{I,\beta}$ *ist für Terme wie folgt induktiv definiert:*

$$\mathrm{eval}_{I,\beta}(x) = \beta(x) \text{ für } x \in \mathcal{X}$$
$$\mathrm{eval}_{I,\beta}(F(T_1, \ldots, T_n)) = F^I\left(\mathrm{eval}_{I,\beta}(T_1), \ldots, \mathrm{eval}_{I,\beta}(T_n)\right)$$

Die Anwendung von $\mathrm{eval}_{I,\beta}$ *auf eine Formel ergibt einen Wahrheitswert nach der üblichen Semantik.*

Wenn die Interpretation I im gegebenen Kontext klar ist, schreiben wir kurz eval_β für $\mathrm{eval}_{I,\beta}$.

Definition 11.5 (Erfüllbarkeit) *Eine Belegung β* erfüllt *eine Formel Φ für eine gegebene Interpretation, wenn die Evaluierungsformel für diese zu* true *auswertet. Eine solche Belegung nennen wir auch* Modell *von Φ. Eine Formel Φ heißt* erfüllbar*, wenn sie mindestens ein Modell hat.*

Terme, die für jede Belegung zum gleichen Wert evaluieren, nennen wir *semantisch äquivalent*.

Definition 11.6 (Termäquivalenz) *Seien T_1, T_2 Terme und I eine Interpretation. T_1 und T_2 heißen* semantisch äquivalent *($T_1 \equiv_I T_2$), wenn* $\mathrm{eval}_{I,\beta}(T_1) = \mathrm{eval}_{I,\beta}(T_2)$ *für jede Belegung β.*

Wenn die Interpretation I im gegebenen Kontext klar ist, schreiben wir kurz $\equiv$ für $\equiv_I$.

Semantisch äquivalente Terme identifizieren wir miteinander. Insbesondere mit Blick auf Lesbarkeit und Verständlichkeit ist diese Identifikation sinnvoll, da sie uns erlaubt, komplizierte Terme durch einfachere äquivalente Terme zu ersetzen.

11.2. Formelautomaten

Wir werden nun eine Klasse von Serviceautomaten definieren, die wir mit Hilfe von Entscheidungsverfahren für Formeln der Prädikatenlogik erster Stufe analysieren können. Einen Serviceautomaten, dessen Guards wir mit Formeln der Prädikatenlogik erster Stufe darstellen können, nennen wir *Formelautomat*.

Definition 11.7 (Formelautomat) *Sei A ein Serviceautomat, so dass es zu jedem Guard G eine prädikatenlogische Formel Φ_G gibt, so dass G die Menge der erfüllenden Belegungen von Φ_G ist. Dann nennen wir A* Formelautomat.

Die in Kapitel 2 allgemein für Serviceautomaten definierten Operationen können im speziellen auch auf Formelautomaten angewendet werden. Für die Operationen, die Prädikate des Serviceautomaten verändern, müssen wir zeigen, dass die resultierenden Prädikate wieder als prädikatenlogische Formeln darstellbar sind. Diese ist aber bei allen vorgestellten Operationen meist durch einfache boolesche Verknüpfungen möglich. Jeder Guard einer Kante des Kreuzproduktes zweier Formelautomaten entsteht z. B. durch Konjunktion zweier Guards der verknüpften Formelautomaten.

Korollar 11.1 *Das Kreuzprodukt zweier Formelautomaten ist wieder ein Formelautomat.*

In den folgenden Beispielen werden wir bevorzugt *Presburger-Arithmetik* [74] als Interpretation für unsere Beispiele verwenden. Die in dieser Arbeit vorgestellten Ergebnisse sind jedoch nicht spezifisch für Presburger-Artithmetik. Jede der in [67] aufgeführten entscheidbaren Theorien wäre eine ebenso geeignete Wahl. Presburger-Arithmetik ist eine Theorie der ganzen Zahlen mit Addition. Auf der Trägermenge $\mathbb{Z}$ sind die Funktionen $+$ und $-$ sowie die Relationen $<, \leq$ und $=$ mit ihrer Standardbedeutung definiert. Ein Beispiel für eine Formel in Presburger-Arithmetik ist z. B.

$$\exists x : 3x + y \leq z \wedge x = y$$

Die Erfüllbarkeit einer Presburger-Formel kann durch *Quantorenelimination* [16] oder automatenbasierte [44] Ansätze entschieden werden. Die Laufzeitkomplexität der Entscheidungsverfahren wächst stark mit der Anzahl der Quantoren in einer Formel [24].

Formelautomaten, deren Formeln in Presburger-Arithmetik darstellbar sind, nennen wir kurz *Presburgerautomaten*. Die Serviceautomaten in Abb. 11.1 und Abb. 11.2 sind Presburgerautomaten.

11.3. Erreichbarkeit und schwache Terminierung

Bevor wir uns der Synthese eines Partners zuwenden, leiten wir in diesem Abschnitt zunächst elementare Formeln zur Beschreibung erreichbarer Zustände eines Formelautomaten her. Diese brauchen wir später bei der Partnersynthese, um unendlich große Mengen guter bzw. schlechter Zustände der Überapproximation in endlicher Form darzustellen.

Ein nützliches Nebenresultat ist, dass wir mit Hilfe dieser Formeln entscheiden können, ob ein Serviceautomat der von uns betrachteten Serviceautomatenklasse schwach terminiert. Damit können wir entscheiden, ob zwei Serviceautomaten Partner sind.

Für jeden Knoten q eines azyklischen Formelautomaten können wir eine Formel angeben, deren Modellmenge genau die erreichbaren Instanzen von q beschreibt. Diese Formel nennen wir die *Erreichbarkeitsformel* von q. Für den Anfangsknoten ist die Erreichbarkeitsformel stets TRUE. Die Erreichbarkeitsformeln aller anderen Knoten können gebildet werden, indem der azyklische Formelautomat beim Anfangsknoten beginnend von oben nach unten abgearbeitet wird. Die Erreichbarkeitsformel eines Knotens q ergibt sich jeweils aus den Erreichbarkeitsformeln seiner direkten Vorgänger und den Guards der Kanten, die q mit diesen verbindet. Die in den Guards vorkommenden new-Variablen bilden wir auf die ihnen zugeordneten Variablen des Knotens q ab. $\text{RENAME}_{\text{new}}$ bezeichne im folgenden die Funktion, die jede Variable x^{new} in einer Formel durch die mit ihr assoziierte Variable x ersetzt.

Definition 11.8 (Erreichbarkeitsformel) *Sei A ein azyklischer Formelautomat mit der Kantenmenge E. Wir definieren die* Erreichbarkeitsformel Φ_q^{Re} *induktiv für jeden Knoten q von A. Für den Anfangsknoten q_0 sei*

$$\Phi_{q_0}^{Re} \underset{Def}{=} \text{TRUE}$$

Für jeden anderen Knoten q sei

$$\Phi_q^{Re} \underset{Def}{=} \bigvee_{e=(q',c,G,q)\in E} \Phi_e^{Re}$$

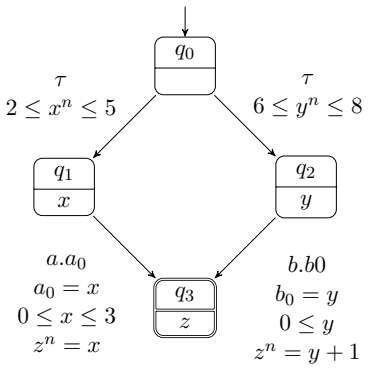
Dabei sei für jede Kante $e = (q', c, G, q)$

$$\Phi_e^{Re} \underset{Def}{=} \text{RENAME}_{\text{new}}\left(\exists x_1, \ldots, x_n : \exists c_1, \ldots, c_m : \Phi_{q'}^{Re} \wedge \Phi_G)\right)$$

wobei $x_1, \ldots, x_n$ die Variablen von q' und $c_1, \ldots, c_m$ die Variablen des Kanals c bezeichnen (zur Erinnerung: Für $c = \tau$ ist die Variablenmenge von c leer).

Korollar 11.2 *Für jeden Zustand (q, β) eines ein Formelautomaten A gilt: β erfüllt Φ_q^{Re} gdw. (q, β) ist erreichbar in A.*

Beweis Beweis durch Induktion.
Anfang: Für q_0 gilt die Behauptung Trivial.



q	Φ_q^{Re}	Φ_q^{Fin}
q_0	TRUE	TRUE
q_1	$2 \leq x \leq 5$	$0 \leq x \leq 3$
q_2	$6 \leq y \leq 8$	$0 \leq y$
q_3	$(2 \leq z \leq 3) \vee (7 \leq z \leq 9)$	TRUE

Abbildung 11.1.: Azyklischer Formelautomat

Schritt: Sei $e = (q', c, G, q) \in E$ eine Kante. Nach Vor. ist (q', β') erreichbar gdw. β' erfüllt $\Phi_{q'}^{Re}$. Nach Konstruktion von Φ_e^{Re} gilt $\beta \in \Phi_{q'}^{Re}$ gdw. es ex. ein erreichbarer Zustand (q', β') mit $(q', \beta') \xrightarrow{e,\gamma} (q, \beta)$. Daraus folgt (q, β) ist erreichbar gdw. β erfüllt Φ_q^{Re}. □

Beispiel Abb. 11.1 zeigt einen Formelautomaten und seine Erreichbarkeits- und Finalformeln. Die Gültigkeit der Formeln $\Phi_{q_0}^{Re}, \Phi_{q_1}^{Re}, \Phi_{q_2}^{Re}$ ist einfach zu sehen. Die Formel $\Phi_{q_3}^{Re}$ ergibt sich aus den Erreichbarkeitsformeln der beiden Vorgängerknoten und den Guards der beiden Kanten. Für die Kante von q_1 zu q_3 erhalten wir die Formel

$$\exists x : \exists a_0 : (2 \leq x \leq 5) \wedge (a_0 = x \wedge 0 \leq x \leq 3 \wedge z^n = x)$$

Dies ist äquivalent zu $2 \leq z^n \leq 3$. Für die Kante von q_2 zu q_3 erhalten wir die Formel

$$\exists y : \exists b_0 : (6 \leq y \leq 8) \wedge (b_0 = y \wedge 0 \leq y \wedge z^n = y + 1)$$

Dies ist äquivalent zu $7 \leq z^n \leq 9$. Durch Umbenennen von z^n in z und disjunktive Verknüpfung der beiden Formeln erhalten wir die Formel $\Phi_{q_3}^{Re}$.

Analog dazu definieren wir für jeden Knoten q eine *Finalformel*, welche die Belegungen von q beschreibt, für die ein Endzustand erreichbar ist. Diese berechnen wir von den Endzuständen aus rückwärts. In der folgenden Definition bezeichne $\text{RENAME}_{\text{old}}$ die Funktion, welche die Variablen in den Formeln der Nachfolger von q durch ihre jeweiligen new-Variablen ersetzt (d. h., jede Variable x wird durch x^{new} ersetzt).

Definition 11.9 (Finalformel) *Sei A ein azyklischer Formelautomat mit Anfangsknoten q_0 und Kantenmenge E. Für jeden Knoten q von A sei die* Finalformel Φ_q^{Fin} *induktiv wie folgt definiert:*

$$\Phi_q^{Fin} \underset{Def}{=} \text{TRUE}$$

falls q ein Endknoten und andernfalls

$$\Phi_q^{Fin} \underset{Def}{=} \bigvee_{e=(q,c,G,q')\in E} \Phi_e^{Fin}$$

Dabei sei für jede Kante $e = (q, c, G, q')$

$$\Phi_e^{Fin} \underset{Def}{=} \exists x_1^{\text{new}}, \ldots, x_n^{\text{new}} : \exists c_1, \ldots, c_m : \text{RENAME}_{\text{old}} \left(\Phi_{q'}^{Fin} \right) \wedge \Phi_G)$$

wobei $x_1, \ldots, x_n$ die Variablen von q' und $c_1, \ldots, c_m$ die Variablen des Kanals c bezeichnen.

Die Finalformel beschreibt die schwächste Vorbedingung dafür, dass ein Endzustand erreicht werden kann und entspricht in etwa der *weakest precondition* in Dijkstras predicate transformer semantics [19].

Korollar 11.3 *Für jeden Zustand (q, β) eines Formelautomaten A gilt:*
β erfüllt Φ_q^{Fin} gdw. von (q, β) ein Endzustand erreichbar ist.

Aus den Korollaren 11.2 und 11.3 folgt unmittelbar, dass genau dann von jeder erreichbaren Instanz von q ein Endzustand erreichbar ist, wenn jede Belegung, die Φ_q^{Re} erfüllt, auch Φ_q^{Fin} erfüllt. Damit können wir durch prädikatenlogische Formeln charakterisieren, wann ein azyklischer Formelautomat schwach terminiert.

Theorem 11.4 (Schwache Terminierung) *Ein azyklischer Formelautomat A terminiert schwach gdw. für jeden Knoten q von A*

$$\forall x_1, \ldots, x_n : \Phi_q^{Re} \implies \Phi_q^{Fin}$$

gilt, wobei $x_1, \ldots, x_n$ die Variablen von q bezeichnen.

Damit können wir insbesondere für azyklische Presburgerautomaten entscheiden, ob diese Partner sind. Dazu müssen wir lediglich Theorem 11.4 auf das Kreuzprodukt der Automaten anwenden.

Beispiel Die Belegung $x = 4$ erfüllt $\Phi_{q_1}^{Re}$, aber nicht $\Phi_{q_1}^{Fin}$. Vom Zustand $(q_1, x = 4)$ ist kein Endzustand erreichbar. Für alle anderen Knoten gilt $\Phi_q^{Re} \implies \Phi_q^{Fin}$.

11.4. Synthesealgorithmus

In diesem Abschnitt stellen wir einen Synthesealgorithmus für azyklische Formelautomaten vor. Die Synthese eines Partners eines Formelautomaten folgt dem selben Prinzip wie die Synthese eines Partners eines Serviceautomaten mit endlich vielen Zuständen, welches wir in Kapitel 10 vorgestellt haben. Zuerst wird ein Formelautomat konstruiert, der zu der in Kapitel 10 definierten Überapproximation der Partnermenge äquivalent ist,

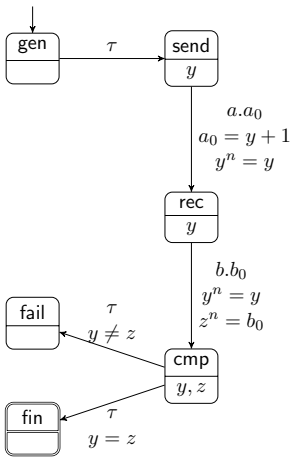


Abbildung 11.2.: Serviceautomat PlusOne

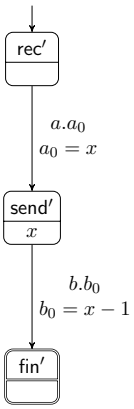


Abbildung 11.3.: Partner von PlusOne

d. h., das gleiche Transitionssystem besitzt. Auf dieser Variante der Überapproximation werden Operationen ausgeführt, welche äquivalent zum Entfernen von Zuständen aus dem Erreichbarkeitsgraphen sind.

Wir stellen zunächst die Darstellung der Überapproximation der Partnermenge aus Def. 10.1 als Formelautomat vor. Danach zeigen wir, wie wir Mengen schlechter Zustände mit Hilfe prädikatenlogischer Formeln darstellen und durch eine geeignete Operation aus der Überapproximation entfernen können. Als Illustration dient uns dabei der Serviceautomat PlusOne in Abb. 11.2. Dieser ist eine Variante des Serviceautomaten AddFunc aus Abb. 5.6 in Kapitel 5.

Im folgenden nehmen wir an, dass ein azyklischer Formelautomat A mit endlich vielen Knoten gegeben ist. Wir definieren einen zur in Def. 10.1 definierten Überapproximation der Partnermenge äquivalenten Formelautomaten, den *Historybaum*. Die Anzahl der Knoten des Historybaumes ist endlich. Der Historybaum kann aber dank der Verwen-

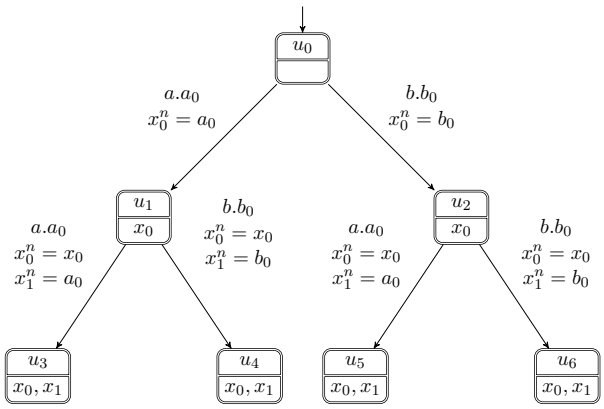


Abbildung 11.4.: Historybaum $\mathsf{U_0}$ der Tiefe 2

dung von Variablen unendlich viele Zustände haben. Die Konstruktion des Historybaumes reflektiert die Beobachtung, dass ein Partner eines Serviceautomaten im allgemeinen alle vom Serviceautomaten kommunizierten Nachrichten speichern muss.

Abb. 11.4 zeigt den zu PlusOne interfaceäquivalenten Historybaum $\mathsf{U_0}$ der Tiefe 2.

Der Historybaum hat für jede Sequenz von Kanälen genau einen Knoten. Jeder Knoten hat Variablen, welche den Inhalt der vom Historybaum kommunizierten Nachrichten speichern. Die Anzahl der Variablen der Knoten wächst mit jeder Nachricht, die der Historybaum kommuniziert. Jedes Eingabewort legt eindeutig einen Knoten und die Belegung seiner Variablen fest. Umgekehrt kann dieses Eingabewort aus der Belegung der Variablen eines Knotens rekonstruiert werden.

Beispiel In $\mathsf{U_0}$ sind z. B. das Eingabewort $a.2\,b.3$ und der Zustand $(u_4, x_0 = 2, x_1 = 3)$ einander eineindeutig zugeordnet.

Der Erreichbarkeitsgraph des Historybaumes ist daher ein deterministischer Baum.

Definition 11.10 (Historybaum) *Sei A ein azyklischer Formelautomat der Tiefe $k \in \mathbb{N}$ mit der Kanalmenge C. Der* Historybaum *(zu A) der Tiefe k ist der Formelautomat $U = (Q, C, var, q_0, E, \Omega)$ mit*

- *der Knotenmenge $Q = C^k$,*
- *dem Anfangsknoten $q_0 = \varepsilon$,*
- *und der Endknotenmenge $\Omega = Q$.*

Die Variablenmenge jedes Knotens sei induktiv für jeden Knoten wie folgt definiert:
Für den Anfangsknoten sei $var(\varepsilon) = \emptyset$.
Für $w \in C^{k-1}$ und $c \in C$ sei $var(wc) = var(w) \cup \{x_{w,c,1}, \ldots, x_{w,c,n}\}$.
Dabei sei n die Anzahl der Variablen des Kanals c und $x_{w,c,i}$ bezeichne für jedes i jeweils

eine frische Variable. Jeder Kanal von U habe die gleichen Variablen wie in A. Die Kantenmenge sei definiert als

$$E \underset{Def}{=} \{(w, c, G_{w,c}, wc) | w \in C^{k-1}, c \in C\}$$

wobei der Guard $G_{w,c}$ einer Kante dargestellt werde durch die Formel

$$\Phi_{G_{w,c}} \underset{Def}{=} \bigwedge_{x \in var(w)} x^{\text{new}} = x \wedge \bigwedge_{1 \leq i \leq n} x^{\text{new}}_{w,c,i} = c_i$$

Dabei sei $var(c) = \{c_1, \ldots, c_n\}$ für $c \in C$.

Die Konstruktion des Historybaumes ist unabhängig von der Wahl des Universums $\mathcal{U}$.

Die Entfaltung des Historybaumes zu einem Formelautomaten A der Tiefe k ist isomorph zum kanonischen Baum mit der Sprache $\Sigma^k(A)$. Dieser ist nach Definition die kanonische Überapproximation der Partnermenge von A.

Korollar 11.5 *Sei A ein azyklischer Formelautomat der Tiefe k. Die Entfaltung des kanonischen Historybaumes zu A der Tiefe k ist isomorph zur kanonischen Überapproximation in Def. 10.1.*

Wir definieren nun eine Operation auf dem Historybaum, die äquivalent zum Entfernen schlechter Zustände aus der Überapproximation ist.

Wir leiten zunächst Formeln her zum Beschreiben von Mengen guter bzw. schlechter Zustände des Historybaumes. Dafür benutzen wir die in Abschnitt 11.3 definierten Erreichbarkeits- und Finalformeln.

Mit Hilfe der Formeln können wir für eine konkrete Belegung der Variablen eines Knotens (p, q) des Kreuzproduktes des gegebenen Serviceautomaten A und der Überapproximation U prüfen, ob von dem durch diese Belegung spezifizierten Zustand ein Endzustand des Kreuzproduktes erreichbar ist. Falls von (p, q) für diese Belegung kein Endzustand erreichbar ist, bildet q zusammen mit dieser Belegung einen schlechten Zustand des Historybaumes.

Diesen Sachverhalt illustrieren wir am Beispiel.

Beispiel Abb. 11.6 zeigt exemplarisch zwei Abläufe (von unendlich vielen) des Kreuzproduktes von PlusOne und dem Historybaum U_0 der Tiefe 2. Vom erreichbaren Zustand $(\mathsf{cmp}, u_4, y = 3, z = 5, x_0 = 4, x_1 = 5)$ des Kreuzproduktes ist kein Endzustand erreichbar. Daraus folgt unmittelbar, dass $(u_4, x_0 = 4, x_1 = 5)$ ein schlechter Zustand von U_0 ist.

Wie man leicht anhand von Tabelle 11.1 überprüfen kann, erfüllt die Belegung $y \mapsto 3, z \mapsto 5, x_0 \mapsto 4, x_1 \mapsto 5$ des Zustands des Kreuzproduktes die Erreichbarkeitsformel des Knotenpaares (cmp, u_4), aber nicht dessen Finalformel.

Aus einer Belegung, welche die Erreichbarkeitsformel erfüllt, aber nicht die Finalformel, erhalten wir also durch Einschränkung auf die Variablen der Überapproximation die Belegung eines schlechten Zustandes der Überapproximation. Diese Beobachtung können wir wie folgt formalisieren.

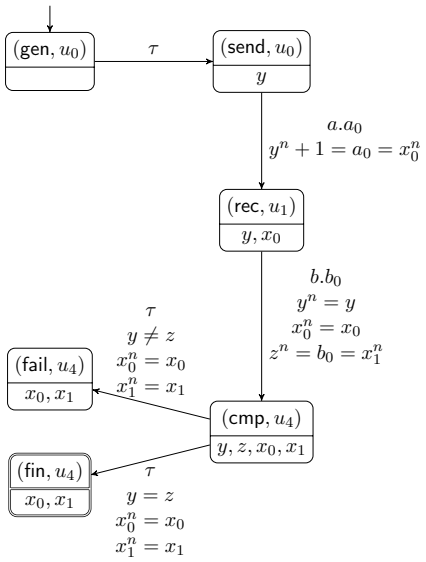


Abbildung 11.5.: Kreuzprodukt PlusOne $\times$ U_0

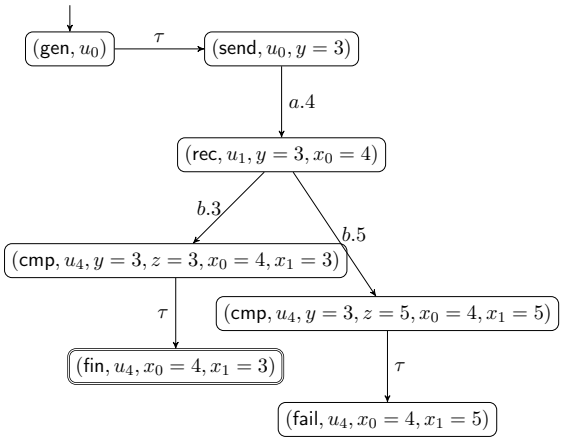


Abbildung 11.6.: Transitionssystem von PlusOne $\times$ U_0 (Ausschnitt)

PlusOne $\times$ U_0		
q	Φ_q^{Re}	Φ_q^{Fin}
(gen, u_0)	TRUE	TRUE
(send, u_0)	TRUE	TRUE
(rec, u_1)	$y + 1 = x_0$	TRUE
(cmp, u_4)	$y + 1 = x_0 \wedge z = x_1$	$y = z$
(fail, u_4)	$x_1 \neq x_0 - 1$	FALSE
(fin, u_4)	$x_1 = x_0 - 1$	TRUE

Tabelle 11.1.: Erreichbarkeits- und Finalformeln für PlusOne $\times$ U_0

Korollar 11.6 *Seien A, B interfaceäquivalente Serviceautomaten, (p, q) ein Knoten von $A \times B$, $var(q)$ die Menge der Variablen von q und γ eine Belegung, die $\Phi^{Re}_{(p,q)}$ erfüllt, aber $\Phi^{Fin}_{(p,q)}$ nicht erfüllt. Dann ist $(q, \gamma_{|var(q)})$ ein schlechter Zustand.*

Um zu entscheiden, ob ein Zustand (q, β) der Überapproximation ein guter oder ein schlechter Zustand ist, müssen wir *alle* Knotenpaare (p, q) und *alle* Belegungen γ überprüfen, die auf den Variablen von q mit β übereinstimmen. Diese Überlegung führt zu einer Charakterisierung der guten Instanzen des Knotens q durch eine Formel Φ^{Good}_q, die wie folgt definiert ist.

Definition 11.11 (Formel für gute Zustände) *Seien A, B kompatible Formelautomaten mit den Knotenmengen Q_A und Q_B und q ein Knoten von B. Die Formel Φ^{Good}_q sei definiert als*

$$\Phi^{Good}_q \underset{Def}{=} \bigwedge_{p \in Q_A} \forall x^p_1, \ldots, x^p_{n_p} : \Phi^{Re}_{(p,q)} \implies \Phi^{Fin}_{(p,q)}$$

Dabei bezeichnen $x^p_1, \ldots, x^p_{n_p}$ jeweils die Variablen des Knotens p.

Die Formel Φ^{Good}_q beschreibt genau die erreichbaren Instanzen eines Knotens q, die gute Zustände sind.

Lemma 11.7 *Seien A, B interfaceäquivalente Formelautomaten. Ein Zustand (q, β) von B ist ein guter Zustand bzgl. A gdw. β erfüllt Φ^{Good}_q.*

Beweis $\implies$: Sei (q, β) ein guter Zustand vom B, p ein Knoten von A und α eine beliebige Belegung von $x_1, \ldots, x_n$. Sei γ eine Belegung, die auf den Variablen von p mit α übereinstimmt und auf den Variablen von q mit β übereinstimmt.
Wir zeigen: γ erfüllt $\Phi^{Re}_{(p,q)} \implies \Phi^{Fin}_{(p,q)}$.
Fall 1: (p, q, γ) ist nicht erreichbar in $A \times B$. Dann ist $\Phi^{Re}_{(p,q)}$ nicht erfüllt und die obige Formel gilt.
Fall 2: (p, q, γ) ist erreichbar in $A \times B$. Nach Def. (guter Zustand) ist von (p, q, γ) ein Endzustand erreichbar. Nach Korollar 11.3 ist $\Phi^{Fin}_{(p,q)}$ erfüllt und die obige Formel gilt.

$\impliedby$: Sei β eine Belegung, die $\forall x_1, \ldots, x_m : \Phi^{Re}_{(p,q)} \implies \Phi^{Fin}_{(p,q)}$ erfüllt.
Wir zeigen: β ist ein guter Zustand. Sei $(p, \alpha) \in \text{know}(q, \beta)$ beliebig. Sei γ eine Belegung, die auf den Variablen von p mit α übereinstimmt und auf den Variablen von q mit β übereinstimmt. Dann erfüllt γ die Formel $\Phi^{Re}_{(p,q)} \implies \Phi^{Fin}_{(p,q)}$. Da (p, q, γ) nach Def. (Wissen) erreichbar ist, erfüllt γ die Formel $\Phi^{Re}_{(p,q)}$. Daraus folgt, dass auch $\Phi^{Fin}_{(p,q)}$ erfüllt ist.
Damit ist von (p, q, γ) ein Endzustand erreichbar. Nach Def. ist (q, α) ein guter Zustand.
□

Die Formel Φ^{Good}_q ist genau dann äquivalent zu TRUE, wenn jede erreichbare Instanz von q ein guter Knoten ist. Zwei Serviceautomaten sind wiederum Partner genau dann, wenn jeder Zustand eines der beiden Serviceautomaten ein guter Zustand ist. Damit erhalten wir den folgenden Satz.

Korollar 11.8 *Seien A und B interfaceäquivalente Formelautomaten. Dann sind A und B Partner gdw.* $\Phi_q^{Good} \equiv \text{TRUE}$ *für jeden Knoten q von B.*

Beispiel Für den Knoten u_4 erhalten wir gemäß Def. 11.11 die Formel

$$\Phi_{u_4}^{Good} = x_1 = x_0 - 1$$

Diese ergibt sich aus den Formeln für die Knotenpaare (cmp, u_4), (fail, u_4), (fin, u_4) in Tabelle 11.1. Man sieht leicht, dass die Formel $x_1 \neq x_0 + 1 \implies$ FALSE (von (fail, u_4)) äquivalent ist zu $x_1 = x_0 + 1$. Ebenso ist $x_1 = x_0 + 1 \implies$ TRUE (von (fin, u_4)) offensichtlich äquivalent zu TRUE.

Für (cmp, u_4) ergibt sich gemäß der Tabelle die Formel $\forall y, z : x_0 = y + 1 \wedge x_1 = z \implies y = z$. Für diese Formel ist die Äquivalenz zu $x_1 = x_0 - 1$ nicht unmittelbar offensichtlich, weswegen hier ein formaler, wenn auch einfacher Beweis notwendig ist.

Korollar 11.9 $\forall y, z : x_0 = y + 1 \wedge x_1 = z \implies y = z$ *ist äquivalent zu* $x_1 = x_0 - 1$.

Beweis $\implies$: Seien x_0, x_1 beliebige Zahlen, so dass die Formel $\forall y, z : x_0 = y+1 \wedge x_1 = z \implies y = z$ gilt. Wähle $y := x_0 - 1$, $z := x_1$. Wegen Gültigkeit der Formel folgt daraus $x_0 - 1 = x_1$.

$\impliedby$: Angenommen, es gilt $x_1 = x_0 - 1$ und $x_0 = y+1 \wedge x_1 = z$. Dann folgt unmittelbar $y = z$. □

Die Belegung des schlechten Zustands $(u_4, x_0 = 4, x_1 = 5)$ erfüllt die Formel $\Phi_{u_4}^{Good}$ nicht. U_0 ist daher offensichtlich kein Partner von PlusOne. Die Belegung des guten Zustands $(u_4, x_0 = 4, x_1 = 3)$ erfüllt dagegen die Formel $\Phi_{u_4}^{Good}$.

Nachdem wir eine Beschreibung der Menge der guten bzw. schlechten Zustände mittels einer prädikatenlogischen Formel definiert haben, definieren wir nun eine Operation auf dem Historybaum, welche die schlechten Zustände aus dem Erreichbarkeitsgraphen der Überapproximation entfernt.

Wir kombinieren dazu die in Def. 2.27 definierte Einschränkungsoperation mit der Formel, welche die guten Zustände des Historybaumes beschreibt. Durch Hinzufügen (d. h., konjunktive Verknüpfung) der Formel Φ_q^{Good} zu jedem Guard einer eingehenden Kante eines Knotens q der Überapproximation schränken wir so die Menge der erreichbaren Instanzen von q auf die guten Zustände ein.

Definition 11.12 (Einschränkung eines Formelautomaten auf gute Zustände) *Seien A ein Formelautomat, q ein Knoten von A, der nicht der Anfangsknoten ist und* Φ *eine Formel, in der nur die Variablen von q frei vorkommen. Sei* A' *der Formelautomat, der aus A entsteht, indem jeder Guard* Φ_e *einer eingehenden Kante e von q durch* $\Phi_e \wedge \text{RENAME}_{\text{old}}(\Phi_q^{Good})$ *ersetzt wird. Dann nennen wir* A' *die* Einschränkung *von A in q auf ihre guten Zustände.*

Der Operator $\text{RENAME}_{\text{old}}$ ersetzt wieder jede Variable x von q durch x^{new}.

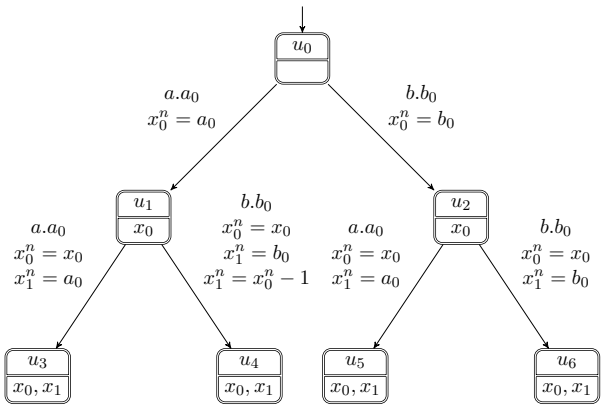


Abbildung 11.7.: Serviceautomat $\mathsf{U_1}$

Beispiel Durch Einschränken von u_4 mit der Formel $x_1 = x_0 + 1$ erhalten wir den Serviceautomaten $\mathsf{U_1}$ in Abb. 11.7. Durch die Einschränkung wird in unserem Beispiel in dem in Abb. 11.6 abgebildeten Transitionssystem die Kante von $(\mathsf{rec}, u_1, y = 3, x_0 = 4)$ zu $(\mathsf{cmp}, u_4, y = 3, z = 5, x_0 = 4, x_1 = 5)$ entfernt, womit der Deadlock $(\mathsf{fail}, u_4, x_0 = 4, x_1 = 5)$ nicht mehr erreichbar ist.

Damit können wir die Synthese eines Partners genau wie in Kapitel 10 durch die Berechnung des Fixpunktes einer Folge von Einschränkungsoperationen beschreiben: Solange ein Knoten q existiert, so dass Φ_q^{Good} nicht äquivalent zu TRUE ist, wenden wir die in Def. 11.12 definierte Einschränkungsoperation auf den Historybaum an. Wenn für ein Folgeglied Φ_q^{Good} äquivalent zu TRUE für jeden Knoten q ist, ist keine weitere Einschränkung möglich. Nach Korollar 11.8 ist das Folgeglied dann ein Partner.

Ist dagegen für ein Folgeglied die Formel $\Phi_{q_0}^{Good}$ für den Anfangsknoten q_0 nicht äquivalent zu TRUE, ist der gegebene Formelautomat nicht bedienbar.

Damit erhalten wir den Algorithmus 2 zur Synthese eines Partners eines azyklischen Formelautomaten. Für jede Formel, welche die guten Instanzen eines Knotens q beschreibt, geben wir dabei das jeweilige Folgeglied, auf die sie sich bezieht, als Superskript an.

Die Korrektheit des Algorithmus folgt daraus, dass der Historybaum und die auf ihm ausgeführte Operation äquivalent sind zu der Überapproximation aus Kapitel 10 und der auf dieser ausgeführten Operation.

Theorem 11.10 (Korrektheit) *Algorithmus 2 gibt genau dann einen Partner von A zurück, wenn A bedienbar ist.*

Beispiel Die Überapproximation $\mathsf{U_1}$ in unserem Beispiel ist nicht weiter einschränkbar. Wie man anhand der in Tab. 11.2 angegebenen Prädikate leicht nachprüfen kann, gilt die in Theroem 11.4 angegebene Implikation für jeden Zustand des Kreuzproduktes. Daher ist auch $\Phi_q^{Good} \equiv$ TRUE für jeden Zustand q von $\mathsf{U_1}$ und $\mathsf{PlusOne} \times \mathsf{U_1}$ terminiert schwach.

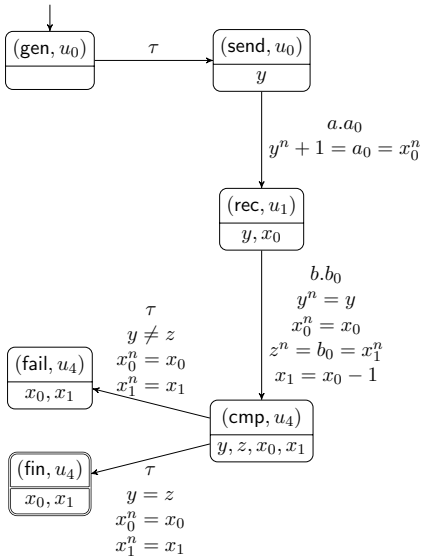


Abbildung 11.8.: Kreuzprodukt PlusOne $\times$ U_1

PlusOne $\times$ U_1		
q	Φ_q^{Re}	Φ_q^{Fin}
(gen, u_0)	TRUE	TRUE
(send, u_0)	TRUE	TRUE
(rec, u_1)	$y + 1 = x_0$	$y + 1 = x_0$
(cmp, u_4)	$y = z = x_1 = x_0 - 1$	$y = z$
(fail, u_4)	FALSE	FALSE
(fin, u_4)	$x_1 = x_0 - 1$	TRUE

Tabelle 11.2.: Erreichbarkeits- und Finalformeln für PlusOne $\times$ U_1

Algorithmus 2 Partnersynthese für azyklische Formelautomaten

Eingabe: Ein azyklischer Formelautomat A der Tiefe $k \in \mathbb{N}$
Bezeichne $U_0 = (Q, C, var, q_0, E, \Omega)$ den kanonischen Historybaum zu A der Tiefe k
$i := 0$
while es ex. $q \in Q \setminus \{q_0\}$ mit $\Phi_q^{Good,U_i} \not\equiv$ TRUE:
    $U_{i+1} :=$ Einschränkung von U_i in q auf gute Zustände von U_i (Def. 11.12)
    $i := i + 1$
if $\Phi_{q_0}^{Good,U_i} \equiv$ TRUE
    return U_i
else
    return „A ist nicht bedienbar“

11.5. Synthese für asynchron kommunizierende Serviceautomaten

Die Synthese eines Partners eines asynchron kommunizierenden azyklischen Formelautomaten kann auf die Synthese eines synchron kommunizierenden azyklischen Formelautomaten zurückgeführt werden. Dazu muss der gegebene Formelautomat lediglich, wie in Abschnitt 10.2 gezeigt, mit einem Nachrichtenpuffer komponiert werden.

Damit die Komposition aus gegebenem Serviceautomaten, Nachrichtenpuffern und Historybaum endlich viele Knoten hat, benötigen wir eine endliche Darstellung des Nachrichtenpuffers.

Abb. 11.9 zeigt einen ungeordneten, unbeschränkten Nachrichtenpuffer für einen Kanal mit einer einzigen Kanalvariablen a_0. Dieser Nachrichtenpuffer ist äquivalent zu dem in in Abschnitt 10.2 beschriebenen Nachrichtenpuffer. Jede Variable speichert den Inhalt genau einer Nachricht. Die Maximalzahl der Nachrichten im Puffer ist nicht begrenzt. Die Inhalte der Nachrichten können in beliebiger Reihenfolge gelesen werden. Nachrichtenpuffer für Kanäle mit mehreren Kanalvariablen werden nach dem gleichen Schema konstruiert.

Die maximale Länge eines Ablaufes der Komposition des gegebenem Serviceautomaten mit den Nachrichtenpuffern und dem Historybaum ist, wie bereits in Abschnitt 10.2 begründet, beschränkt. Die synchrone Komposition des gegebenen Serviceautomaten, seiner Nachrichtenpuffer und des Historybaumes hat daher stets endlich viele Knoten und der Algorithmus 2 kann in der gleichen Weise wie Alorithmus 1 zur Synthese eines Partners verwendet werden. Die Korrektheit wird wieder von Lemma 10.7 garantiert.

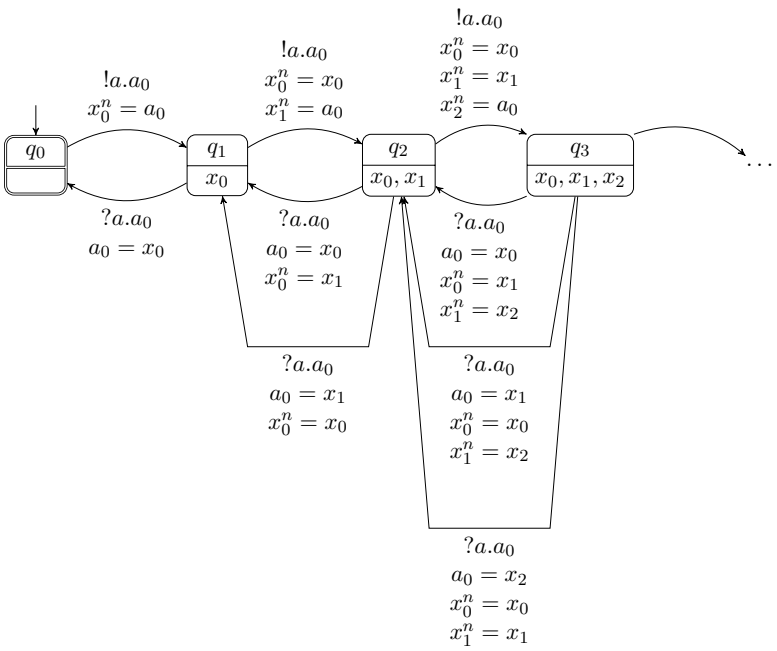


Abbildung 11.9.: Unbeschränkter ungeordneter Nachrichtenpuffer für einen Kanal a

12. Adaptersynthese

Da Services in der Regel unabhängig voneinander entwickelt werden, sind ihre Interfaces häufig nicht miteinander kompatibel. Die Interfaces unterscheiden sich beispielsweise bezüglich der Anzahl der Kanäle oder dem Format der über diese kommunizierten Daten. Die Services können in diesem Fall nicht direkt miteinander komponiert werden. Um zwei Services zu komponieren, deren Interface sich nur in Details unterscheiden, kann ein *Adapter* als Vermittler zwischen den beiden Services dienen.

Beispiel Abb. 12.1 zeigt zwei Serviceautomaten Customer und Shop. Deren Interfaces sind nicht miteinander kompatibel.

Die Menge PRODID bezeichnet eine Menge von Bestellnummern für Produkte und PRICE die Menge der Preise der Produkte. Die Funktion product ordnet jeder Bestellnummer ein Produkt zu. Die Funktionen price und shipping ordnen jeder Bestellnummer den Preis des jeweiligen Produktes bzw. dessen Lieferkosten zu.

Das Interface der Serviceautomaten unterscheidet sich in der Form der Rechnung. Shop schlüsselt die Lieferkosten und den Preis des bestellten Produktes einzeln auf (invoiceLong). Der Serviceautomat Customer dagegen verlangt den Gesamtpreis als Einzelposten (invoiceShort).

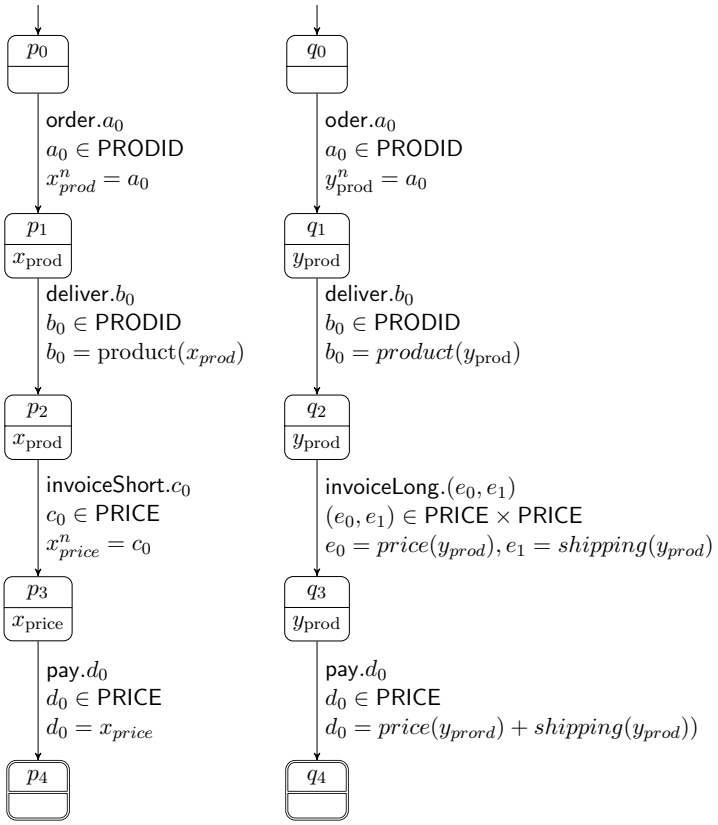
Die beiden Serviceautomaten können aufgrund ihres unterschiedlichen Interfaces nicht direkt miteinander komponiert werden. Sie können aber unter Zuhilfenahme des Adapters in Abb. 12.2 miteinander komponiert werden. Der Adapter übersetzt das Rechnungsformat für den Kunden. Der Adapter addiert dabei die Produkt- und Lieferkosten. Der Kunde erfährt somit die Gesamtkosten des bestellten Produktes und kann diese, wie vom Händler erwartet, bezahlen (pay).

Aufgabe eines Adapters ist es, die Kommunikation zwischen Services so anzupassen, dass das komponierte System eine bestimmte Eigenschaft erfüllt, etwa eines der in Abschnitt 5 eingeführten Kompatibilitätskriterien.

Formal besteht das Problem der Adaptersynthese darin, zu zwei Serviceautomaten A und B einen Serviceautomaten C, den Adapter, zu finden, so dass die Komposition von A, C und B das Kompatibilitätskriterium erfüllt. Aus technischen Gründen ist es (abweichend vom Beispiel) dabei sinnvoll, anzunehmen, dass die Interfaces von A und B disjunkt sind. Der Adapter C ist also ein Partner der parallelen Komposition von A und B für das jeweilige Kompatibilitätskriterium.

Das Problem der Adaptersynthese kann daher auf die Partnersynthese zurückgeführt werden. Den in Abschnitt 11.4 vorgestellten Algorithmus können wir daher verwenden, um einen Adapter für azyklische Serviceautomaten zu konstruieren.

Das Verhalten eines auf diese Weise erzeugten Adapters mag bei praktischen Anwendungen nicht immer den Anforderungen genügen. Bestimmte Nachrichten wie Passwörter



(a) Customer (b) Shop

Abbildung 12.1.: Nicht miteinander kompatible Serviceautomaten

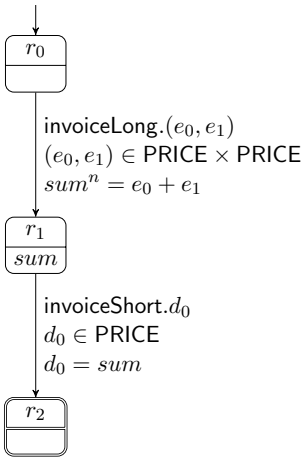


Abbildung 12.2.: Adapter für Customer und Shop

oder Geld können durch den Adapter aus dem Nichts erschaffen werden, um die Services in ihre Endzustände zu bringen.

Damit ein Adapter für praktische Anwendungen tauglich ist, sind weitere Beschränkungen der Fähigkeiten des Adapters notwendig.

Gierds [27] stellt einen petrinetzbasierten Ansatz zur Adaptersynthese vor. Dieser erlaubt, durch eine Menge von Transformationsregeln zu spezifizieren, welche Nachrichten neu erzeugt oder ineinander umgewandelt werden dürfen und welche nicht. Die Einhaltung der Transformationregeln wird dadurch erzwungen, dass ein weiterer Service E der Komposition hinzugefügt wird, welcher die durch die Transformationsregeln vorgegebenen Beschränkungen erzwingt.

Über eine zu diesen Petrinetzen analoge Konstruktion wäre auch in unserem Formalismus möglich. Diese müssten aber an die derzeitigen Beschränkungen unseres Formalismus angepasst werden und dürften beispielsweise keine Zyklen enthalten. Eine sinnvolle Umsetzung einer der Synthese eines Adapters bleibt daher ein offenes Problem.

Fahland et al. [21] beschreiben eine Anwendung der Adaptersynthese auf Servicemodelle mit endlich vielen Zuständen, die mit CPN Tools erstellt wurden. Dabei stehen nicht datenspezifische Aspekte der Kommunikation wie die Reihenfolge der Nachrichten im Vordergrund.

Eine offene Frage bleibt, wie die Transformationsregeln um datenspezifische Aspekte erweitert werden könnten. Denkbar wäre z. B., eine Obergrenze für Kredite angeben zu können, welche vorübergehend geschaffen oder geliehen werden können.

13. Partnersynthese für Identitätsautomaten

In diesem Kapitel stellen wir einen weiteren Synthesealgorithmus speziell für Partner azyklischer Identitätsautomaten vor. Da Identitätsautomaten ein Spezialfall von Formelautomaten sind, können wir Partner azyklischer Identitätsautomaten bereits mit dem in Kapitel 11 vorgestellten Algorithmus konstruieren. Wir benötigen den neuen Synthesealgorithmus daher nicht, um die Bedienbarkeit eines Identitätsautomaten zu entscheiden.

Ziel dieses Kapitels ist es vielmehr, eine Analysetechnik speziell für Identitätsautomaten vorzustellen. Wir stellen einen graphbasierten Synthesealgorithmus für Partner vor. Die Konzepte, die wir in Kapitel 10 für einzelne Zustände definiert haben, verallgemeinern wir auf Mengen von Zuständen, welche die Knoten eines Graphen bilden. Das Verhalten eines Service spiegelt sich in der Struktur dieses Graphen wieder. Dies erleichtert es, sich wiederholendes Verhalten des Service zu erkennen und Techniken wie Faltung auf den Service anzuwenden.

Wir glauben, dass der in diesem Kapitel vorgestellte graphbasierte Ansatz geeignet ist, insbesondere Partner zyklischer Identitätsautomaten zu analysieren und zu synthetisieren. Zwar hat nicht jeder Identitätsautomat einen endlich darstellbaren Partner. Es wäre jedoch wünschenswert, einen Algorithmus angeben zu können, der einen Partner eines zyklischen Identitätsautomaten synthetisieren kann, falls ein endlich darstellbarer Partner existiert.

Ein solcher Algorithmus zur Synthese eines Partners eines zyklischen Serviceautomaten konnte in dieser Arbeit nicht gefunden werden. Die in diesem Kapitel vorgestellte Darstellung des Zustandsraumes eines Serviceautomaten bildet jedoch eine gute Grundlage, um graphbasierte Techniken zu untersuchen, die die bei der Synthese eines zyklischen Partner hilfreich sein könnten. Durch Faltung der graphbasierten Darstellung des Zustandsraumes könnte es möglich sein, zyklische Partner zu erzeugen.

Die Knoten des Graphen, der den Zustandsraum eines Identitätsautomaten darstellt, definieren wir als Äquivalenzklassen von Zuständen. Äquivalente Zustände verhalten sich gleich bezüglich aller für den Algorithmus relevanten Eigenschaften. Die Definition der Äquivalenzklassen nutzt Symmetrien zwischen Abläufen von Identitätsautomaten aus. Die ausgenutzten Symmetrien gelten übergreifend auf allen Identitätsautomaten. Die Äquivalenzen der Zustände bleiben daher erhalten, wenn der Identitätsautomat verändert wird, etwa durch Entfernen von Knoten oder durch Komposition mit einem anderen Identitätsautomaten.

Der in diesem Kapitel vorgestellte Synthesealgorithmus konstruiert zwei Graphen: Einen Graphen für die Überapproximation der Partner des gegebenen Serviceautomaten, aus welcher durch Entfernen von Knoten ein Partner erzeugt wird, und einen Graphen

für die Komposition der Überapproximation mit dem gegebenen Serviceautomaten. Die relevanten Eigenschaften äquivalenter Zustände bleiben über alle Iterationsschritte des Algorithmus erhalten. Die Knoten der beiden Graphen müssen daher nicht nach jedem Iterationsschritt neu berechnet werden.

Den Begriff des Wissens eines Zustandes können wir verallgemeinern auf Knoten des Graphen der Überapproximation der Partner eines Identitätsautomaten. Das Wissen jedes Knotens des Graphen der Überapproximation ist eine Menge von Knoten des Graphen der Komposition. Die Verallgemeinerung des Wissens auf Knoten erlaubt wiederum, dass Theorem zum Verschmelzen einer Zustände eines Partners aus Abschnitt 5.3 auf Knoten von Identitätsautomaten zu verallgemeinern.

Für beliebige Formelautomaten konnte in dieser Arbeit keine Verallgemeinerung des Wissens eines Zustandes dieser Art gefunden werden. Die Instanzen der Knoten der in Kapitel 11 betrachteten Formelautomaten verhalten sich nicht bezüglich aller relevanten Eigenschaften gleich.

Wir stellen zunächst den *symbolischen Erreichbarkeitsgraphen* eines Identitätsautomaten vor. Dieser erlaubt auf einfache Weise, die Erreichbarkeit von Zuständen des Identitätsautomaten zu prüfen. Danach zeigen wir, wie der symbolische Erreichbarkeitsgraph verwendet werden kann, um aus einer Variante der Überapproximation der Partnermenge eines azyklischen Identitätsautomaten einen Partner zu konstruieren.

13.1. Symbolischer Erreichbarkeitsgraph

Der symbolische Erreichbarkeitsgraph (SRG) ist eine endliche Darstellung des Transitionssystems eines Identitätsautomaten. Mit dem SRG können wir

- entscheiden, ob ein Zustand erreichbar ist
- entscheiden, ob von einem Zustand aus ein Endzustand erreichbar ist

Wir können dagegen *nicht* für beliebige Paare von Zuständen entscheiden, ob zwischen diesen ein Pfad existiert.

Die Darstellung nutzt Symmetriebeziehungen zwischen Zuständen eines Identitätsautomaten, um Äquivalenzklassen von Zuständen zu bilden. Abb. 13.2 zeigt einen Teil des Erreichbarkeitsgraphen des Identitätsautomaten A aus Abb. 13.1. Aus jeder Äquivalenzklasse sind exemplarisch jeweils nur zwei Zustände dargestellt.

Eine *Symmetrie* ist formal ein *Graphenautomorphismus*, d. h. ein Isomorphismus zwischen einem Graphen und sich selbst. Der Isomorphismus bildet jeden Pfad des Graphen auf einen sich spiegelbildlich zu diesem verhaltenden Pfad ab.

Der in der Arbeit vorgestellte symbolische Erreichbarkeitsgraph ist eine stark vereinfachte Variante des symbolischen Erreichbarkeitsgraphen eines *Well-Formed Coloured Nets* (WFN) [15]. Well-Formed Coloured Nets sind eine Klasse gefärbter Petrinetze, die nur bestimmte Guards und Funktionen erlaubt. Die Knoten des symbolischen Erreichbarkeitsgraphen eines WFN sind Äquivalenzklassen.

Die in dieser Arbeit zur Definition des symbolischen Erreichbarkeitsgraphen verwendete Äquivalenz ist die kanonische Erweiterung der bereits in Abschnitt 5.2 vorgestellten

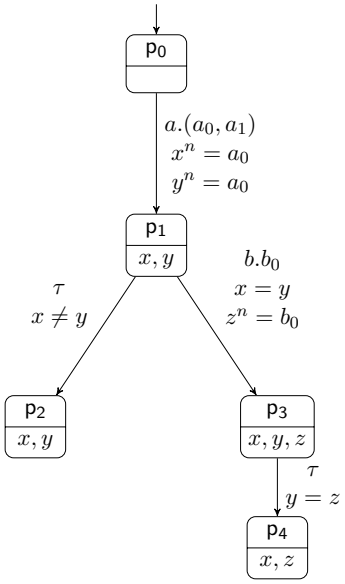


Abbildung 13.1.: Identitätsautomat A

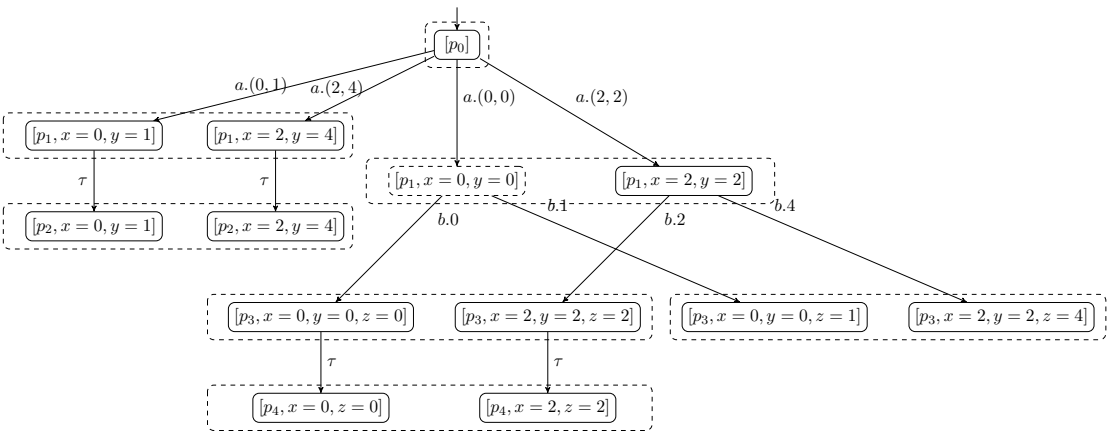


Abbildung 13.2.: Erreichbarkeitsgraph von A (Ausschnitt). Umrandungen kennzeichnen identitätsäquivalente Zustände.

Identitätsäquivalenz (Def. 5.6) auf Zustände und ein Spezialfall der für WFN benutzten Äquivalenz.

Zwei Instanzen eines Knotens heißen *identitätsäquivalent*, wenn jedes Paar von Variablen x und y, die in einer Instanz des Knotens mit dem gleichen Wert belegt sind, auch in der anderen Instanz mit dem gleichen Wert belegt sind.

Definition 13.1 (Identitätsäquivalente Zustände) *Sei q ein Knoten eines Serviceautomaten A. Instanzen $(q, \beta), (q, \beta')$ von q heißen* identitätsäquivalent *(wir schreiben $(q, \beta) \sim_{id} (q, \beta')$), wenn β und β' identitätsäquivalent sind. Die Identitätsäquivalenzklasse eines Zustandes z notieren wir mit $[z]_{id}$.*

Identitätsäquivalente Zustände verhalten sich *symmetrisch* zueinander: Eine Kante eines Identitätsautomaten ist immer entweder für jeden Zustand einer Äquivalenzklasse aktiviert oder für gar keinen. Die Nachfolger identitätsäquivalenter Zustände sind ebenfalls jeweils wieder identitätsäquivalent zueinander. Alle Zustände einer Äquivalenzklasse verhalten sich daher gleich bezüglich Erreichbarkeit vom Anfangszustand.

Die Symmetrien zwischen Zuständen und Abläufen können wir formal durch *Permutationen* beschreiben. Eine Permutation ist eine bijektive Funktion $\sigma : \mathcal{U} \to \mathcal{U}$, die jedem Wert aus dem Universum $\mathcal{U}$ eineindeutig einen anderen Wert zuweist. Permutationen werden häufig zum Beschreiben von Symmetrien in Petrinetzen verwendet [37, 39, 30].

Durch Permutieren der Werte, die in einem Ablauf eines Identitätsautomaten vorkommen, erhalten wir einen zu diesem symmetrischen Ablauf, der ebenfalls ausführbar ist.

Definition 13.2 (Permutieren eines Zustandes) *Sei $z = (q, \beta)$ ein Zustand eines Serviceautomaten A und $\sigma : \mathcal{U} \to \mathcal{U}$ eine Permutation. Dann sei $\sigma(z) \underset{Def}{=} (q, \beta \circ \sigma)$ der Zustand, der aus z durch Anwenden der Permutation σ hervorgeht.*

Belegungen eines Knotens sind genau dann identitätsäquivalent, wenn sie durch Permutation auseinander hervorgehen. Die Permutation von Werten eines Zustandes erhält die paarweisen Gleichheits- bzw. Ungleichheitsbeziegungen zwischen den Werten.

Korollar 13.1 (Symmetrie identitätsäquivalenter Belegungen) *Belegungen β, β' sind identitätsäquivalent gdw. es eine Permutation $\sigma : \mathcal{U} \to \mathcal{U}$ gibt, so dass $\beta = \beta \circ \sigma$.*

Beweis Folgt aus der Beobachtung, dass $u = v$ gdw. $\sigma(u) = \sigma(v)$ für jedes Paar $u, v \in \mathcal{U}$. □

Beispiel Durch Anwenden der Permutation $\sigma : 0 \mapsto 2, 1 \mapsto 4$ geht aus dem Zustand $(p_1, x = 0, y = 1)$ der Zustand $(p_1, x = 2, y = 4)$ hervor. Der Zustand $(p_1, x = 0, y = 0)$ kann dagegen nicht durch Permutation aus $(p_1, x = 0, y = 1)$ erzeugt werden.

Von Zuständen, die durch Anwenden einer Permutation σ auseinander hervorgehen, sind wieder Zustände erreichbar, die durch Anwenden der gleichen Permutation auseinander hervorgehen. Wir können daher eine Permutation auf einen Ablauf anwenden und erhalten wieder einen (zu diesem spiegelbildlichen) Ablauf des Serviceautomaten.

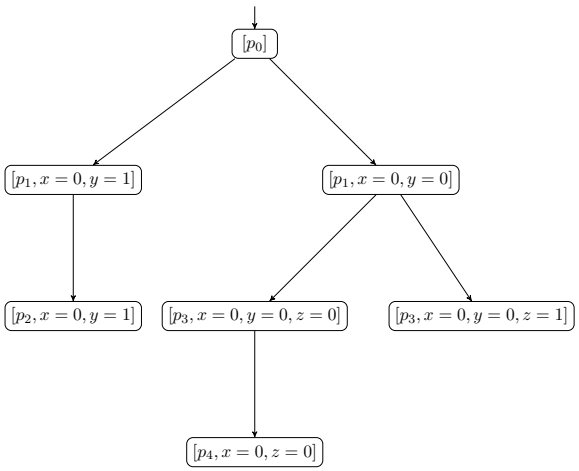


Abbildung 13.3.: Symbolischer Erreichbarkeitsgraph von A

Definition 13.3 (Permutieren eines Eingabewortes) *Sei A ein Identitätsautomat und $\sigma : \mathcal{U} \to \mathcal{U}$ eine Permutation. Für jede Aktion $a = (c, \gamma) \in \Sigma(A)$ bezeichne $\sigma(a)$ die Aktion $(c, \gamma \circ \sigma)$.*

Für $w = a_1 a_2 \dots a_m \in \Sigma^m(A)$ sei $\sigma(w) \underset{Def}{=} \sigma(a_1)\sigma(a_2)\dots\sigma(a_m)$ die Permutation *von w mit σ.*

Beispiel Aus dem Ablauf $p_0 \overset{a.(0,0)}{\to} (p_1, x = 0, y = 1) \overset{b.1}{\to} (p_3, x = 0, y = 0, z = 1)$ wird durch Anwenden der Permutation $\sigma : 0 \mapsto 2, 1 \mapsto 4$ der Ablauf
$p_0 \overset{a.(2,2)}{\to} (p_1, x = 2, y = 4) \overset{b.4}{\to} (p_3, x = 2, y = 2, z = 4)$.

Durch Anwenden einer Permutation auf einen Ablauf erhalten wir einen zu diesem symmetrischen Ablauf, der ebenfalls ausführbar ist.

Korollar 13.2 (Symmetrische Abläufe) *Sei A ein Identitätsautomat und $(q, \beta), (q', \beta')$ Zustände von A und $\sigma : \mathcal{U} \to \mathcal{U}$ eine Permutation.*
Dann gilt $(q, \beta) \overset{w}{\Longrightarrow}_A (q', \beta')$ gdw. $(q, \beta \circ \sigma) \overset{\sigma(w)}{\Longrightarrow}_A (q', \beta' \circ \sigma)$.

Im speziellen folgt daraus, dass entweder jeder Zustand einer Äquivalenzklasse vom Anfangszustand aus erreichbar ist oder gar keiner.

Korollar 13.3 (Symmetrie der Erreichbarkeit) *Sei A ein Identitätsautomat, $z, z' \in Z(A)$ identitätsäquivalente Zustände. Dann ist z erreichbar in A gdw. z' ist erreichbar in A.*

Diese Beobachtung motiviert die Definition des *symbolischen Erreichbarkeitsgraphen* (SRG). Der symbolische Erreichbarkeitsgraph entsteht aus dem Erreichbarkeitsgraphen eines Serviceautomaten durch Verschmelzen der Zustände der Identitätsäquivalenzklassen. Abb. 13.3 zeigt den symbolischen Erreichbarkeitsgraphen des Identitätsautomaten A in Abb. 13.1.

Definition 13.4 (Symbolischer Erreichbarkeitsgraph) *Sei A ein Identitätsautomat. Der symbolische Erreichbarkeitsgraph (SRG) von A ist der Graph mit der Knotenmenge $V = Z(A)\backslash_{\sim_{id}}$*
und der Kantenmenge $E = \left\{([z]_{id}, [z']_{id}) | z \xrightarrow{a}_A z'\right\}$.

Die Äquivalenzklasse des Anfangszustandes bezeichnen wir als den Anfangsknoten *des SRG. Eine Klasse von Instanzen eines Endknotens von A bezeichnen wir ebenso als* Endknoten *des SRG.*

Der symbolische Erreichbarkeitsgraph eines Identitätsautomaten mit endlich vielen Knoten hat ebenfalls endlich viele Knoten: Die Menge der Instanzen eines Knotens q des Identitätsautomaten mit n Variablen zerfällt in höchstens $!n$ Identitätsäquivalenzklassen.

Aus dem symbolischen Erreichbarkeitsgraphen eines Serviceautomaten kann der ursprüngliche Erreichbarkeitsgraph des Serviceautomaten nicht rekonstruiert werden, da die Beschriftungen der Kanten nicht erhalten bleiben. Wir können jedoch Aussagen über die Existenz gewisser Abläufe des Serviceautomaten machen:

Wenn im symbolischen Erreichbarkeitsgraphen eine Kante von einer Klasse Z zu einer Klasse Z' existiert, dann gibt es für jeden Zustand $z \in Z$ des Serviceautomaten einen Schritt zu einem Zustand $z' \in Z'$. Insbesondere können wir aus dem SRG für jeden Zustand des Serviceautomaten erkennen, ob von diesem aus ein Endzustand erreichbar ist.

Theorem 13.4 *Es ist entscheidbar, ob ein Identitätsautomat mit endlich vielen Knoten schwach terminiert.*

Da das Kreuzprodukt zweier Identitätsautomaten wieder ein Identitätsautomat ist, können wir entscheiden, ob diese füreinander Partner sind.

Das Konzept des SRG kann prinzipiell erweitert werden auf Serviceautomaten, die ausdrucksstärker sind als Identitätsautomaten. Die Knoten des ursprünglichen symbolischen Erreichbarkeitsgraphen eines Well-Formed Nets sind Klassen einer permutationsbasierten Äquivalenzrelation, die wesentlich feiner ist als die hier verwendete Identitätsäquivalenz. Der Guard einer Transition eines WFN kann im Unterschied zum Guard eines Identitätsautomaten eine Nachfolgerfunktion enthalten. Mit der feineren Äquivalenzrelation können im SRG Nachfolgerbeziehungen zwischen Werten dargestellt werden. Da die feinere Äquivalenzrelation jedoch zu mehr Äquivalenzklassen führt, kann sie nicht in Kombination mit dem im nächsten Abschnitt vorgestellten Partnersynthesealgorithmus verwendet werden. Der SRG der Überapproximation der Partermenge hätte bei Verwendung der feineren Äquivalenzrelation unendlich viele Knoten.

Im folgenden Abschnitt beschreiben wir die Synthese eines Partners eines azyklischen Identitätsautomaten.

13.2. Synthesealgorithmus

Die Synthese eines Partners eines azyklischen Identitätsautomaten folgt dem gleichen Schema wie in den Kapiteln 10 und 11. Zunächst wird ein zum gegebenen Serviceauto-

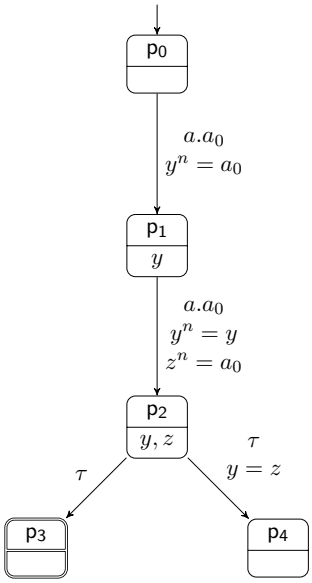


Abbildung 13.4.: Identitätsautomat FailOnEq

maten interfaceäquivalenter Serviceautomat konstruiert, der äquivalent ist zur Überapproximation der Partnermenge aus Kapitel 10. Aus diesem wird dann durch Entfernen von Zuständen ein Partner erzeugt.

Wir definieren zunächst den *Identitätshistorybaum*. Der Identitätshistorybaum ist eine Variante des in Abschnitt 11.4 vorgestellten Historybaumes. Der Erreichbarkeitsgraph des Identitätshistorybaumes ist ebenso wie der Erreichbarkeitsgraph des Historybaumes isomorph zum kanonischen Baum und damit zur kanonischen Überapproximation der Partnermenge in Kapitel 10.

Der Identitätshistoryautomat ist selbst ein Identitätsautomat. Einen Partner eines Identitätsautomaten erzeugen wir durch Entfernen von Knoten aus dem Identitätshistorybaum.

Abb. 13.4 zeigt einen Identitätsautomaten FailOnEq und Abb. 13.6 den zu FailOnEq interfaceäquivalenten Identitätshistorybaum U_0. Der Identitätshistorybaum ist so konstruiert, dass seine Struktur isomorph zu seinem symbolischen Erreichbarkeitsgraphen ist. Dies ermöglicht es später, gezielt Knoten aus dem symbolischen Erreichbarkeitsgraphen zu entfernen.

Der Identitätshistorybaum funktioniert nach dem gleichen Prinzip wie der Historybaum nach Def. 11.10. Der Identitätshistorybaum speichert jeden Wert, der in einem Eingabewort vorkommt, in einer Variablen. Kommt im Eingabewort mehrfach der gleiche Wert vor, wird dieser nur einmal gespeichert. Je nachdem, in welcher Relation die Aktionen des Eingabewortes zueinander stehen, wird in andere Knoten verzweigt. Unterschieden wird dabei nur die paarweise Gleichheit bzw. Ungleichheit von Werten.

In jedem Schritt wird der Inhalt der kommunizierten Nachricht mit jedem bereits in einer Variablen gespeicherten Wert verglichen. Abhängig vom Ergebnis des Vergleichs

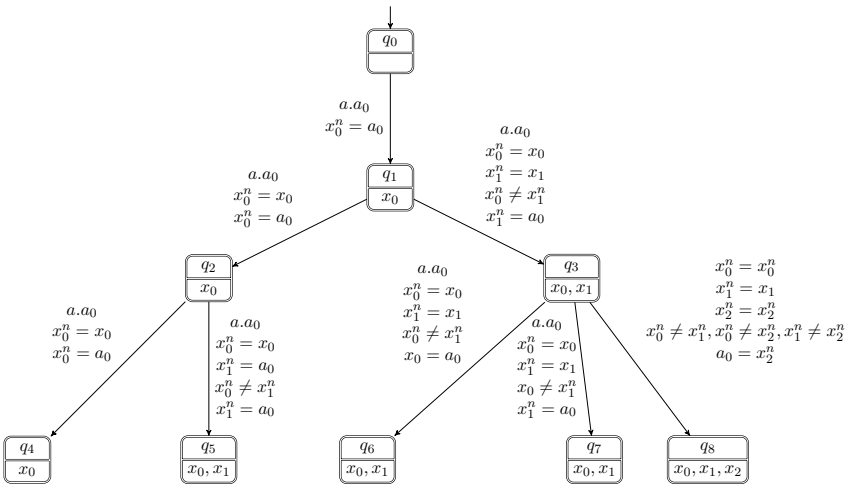


Abbildung 13.5.: Identitätshistorybaum U_0 der Tiefe 3

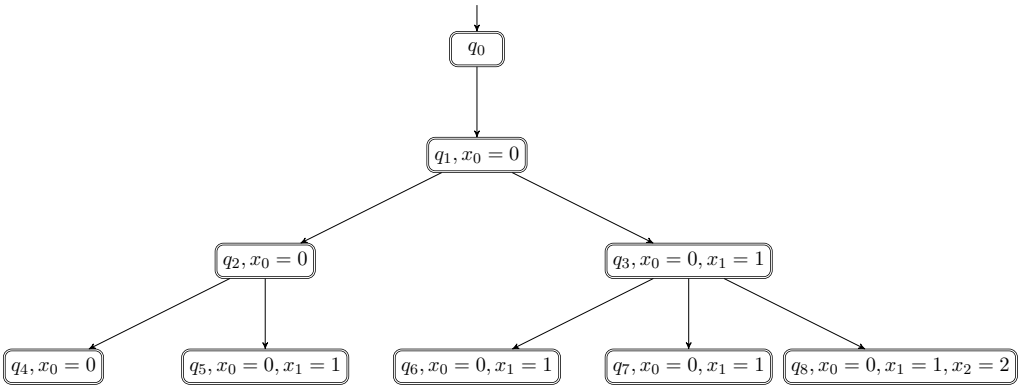


Abbildung 13.6.: Symbolischer Erreichbarkeitsgraph von U_0

trifft der Identitätshistorybaum eine Fallunterscheidung und verzweigt in einen anderen Nachfolgeknoten. Ist der Inhalt der Nachricht noch in keiner Variablen gespeichert, wird in einen Nachfolgeknoten verzweigt, in dem eine neue Variable eingeführt wird, welche den Inhalt der Nachricht speichert. Ist dagegen der Inhalt der Nachricht bereits in einer Variablen gespeichert, wird im Nachfolgeknoten keine neue Variable eingeführt. Je nachdem, mit welcher Variablen der Inhalt der Nachricht übereinstimmt, wird dabei in einen anderen Nachfolgeknoten verzweigt.

Wir beschreiben im Folgenden die Konstruktion des Identitätshistorybaumes. Der Leser kann die formalen Details der Konstruktion bei Bedarf überspringen. Die wesentliche Eigenschaften des Identitätshistorybaumes sind, dass sein Erreichbarkeitsgraph ein kanonischer Baum ist und seine Struktur isomorph zu seinem symbolischen Erreichbarkeitgraphen ist.

Formal definieren wir die Knoten des Identitätshistorybaumes als Äquivalenzklassen

von Eingabeworten. Die Konstruktion des Identitätshistorybaumes über Äquivalenzklassen ist wesentlich einfacher zu definieren als über eine induktive Definition. Die Kanten zwischen den Äquivalenzklassen sind so definiert, dass die Eingabe eines Wortes genau in die Äquivalenzklasse des wortes führt.

Die Äquivalenzklassen werden durch die Identitätsäquivalenzrelation für Eingabeworte gebildet. Zwei Eingabeworte heißen *identitätsäquivalent*, wenn sie durch Anwendung einer Permutation auseinander hervorgehen. Permutieren von Werten erhält analog zu Korollar 13.1 die Gleichheitsbeziehungen zwischen den Werten eines Eingabewortes.

Definition 13.5 (Identitätsäquivalente Einagbeworte) *Sei A ein Identitätsautomat. Worte $w, w' \in \Sigma^n(A)$ heißen* identitätsäquivalent *(geschrieben $w \sim_{id} w'$), wenn es eine Permutation $\sigma : \mathcal{U} \to \mathcal{U}$ gibt mit $\sigma(w) = w'$. Die Identitätsäquivalenzklasse eines Wortes w notieren wir mit $[w]_{id}$.*

Beispiel Das Eingabewort $a.2\, a.2\, a.0$ ist identitätsäquivalent zu $a.0\, a.0\, a.1$. Die Knoten des Identitätshistorybaumes U_0 in Abb. 13.6 sind die folgenden Klassen der Identitätsäquivalenz:

$$\begin{array}{ll} q_0 = [\varepsilon]_{id} & q_5 = [a.0\, a.0\, a.1]_{id} \\ q_1 = [a.0]_{id} & q_6 = [a.0\, a.1\, a.0]_{id} \\ q_2 = [a.0\, a.0]_{id} & q_7 = [a.0\, a.1\, a.1]_{id} \\ q_3 = [a.0\, a.1]_{id} & q_8 = [a.0\, a.1\, a.2]_{id} \\ q_4 = [a.0\, a.0\, a.0]_{id} & \end{array}$$

Die Eingabe $a.0$ $a.0$ $a.0$ führt z. B zu dem Zustand $(q_4, x_0 = 0)$. Die Eingabe $a.0$ $a.1$ $a.2$ führt zu dem Zustand $(q_8, x_0 = 0, x_1 = 1, x_2 = 2)$.

Die Anzahl der Variablen eines Knotens des Identitätshistorybaumes ist gleich der Anzahl der verschiedenen Werte, die in seinem Repräsentanten vorkommen.

Beispiel Der Knoten q_4 mit dem Repräsentanten $a.0\, a.0\, a.0$ hat nur eine Variable x_0. Der Knoten q_8 mit dem Repräsentanten $a.0\, a.1\, a.2$ hat drei Variablen.

Die Anzahl der Variablen ist von der Wahl des Repräsentanten unabhängig.

Um die Kanten zwischen den Knoten des Identitätshistorybaumes zu definieren, benötigen wir für jeden Knoten einen fest gewählten Repräsentanten. Die Repräsentanten der Knoten müssen dabei zueinander passend gewählt werden: Wir verlangen, dass für jeden Repräsentanten w einer Äquivalenzklasse auch jeder Präfix von w der Repräsentant seiner jeweiligen Klasse ist. D. h., den Repräsentanten jedes Knotens wählen wir so, dass er aus dem Repräsentanten seines Vorgängers durch Anhängen einer Aktion hervorgeht.

Beispiel Die Repräsentanten $a.0$ bzw. $a.0\, a.1$ der Knoten q_1 bzw. q_3 von U_0 sind Präfixe des Repräsentanten $a.0\, a.1\, a.2$ von q_8.

Im folgenden bezeichne w_q den fest gewählten Repräsentanten eines Knotens q des Identitätshistorybaumes. Jedem Wert $u \in \mathcal{U}$, der im Repräsentanten w_q eines Knotens q vorkommt, ordnen wir eineindeutig eine Variable x_u zu. Diese Variablen bilden zusammen die Variablenmenge von q.

Beispiel Im Repräsentanten $a.0\,a.1\,a.0$ des Knotens q_6 kommen die Werte 0 und 1 vor. Der Knoten q_6 hat dementsprechend die Variablenmenge x_0 und x_1.

Die Guards der Kanten des Identitätshistorybaumes definieren wir so, dass jede Variable x_u von q mit u belegt wird, wenn der Ablauf mit dem Trace w_q ausgeführt wird. Wird ein Ablauf mit einem zu w_q identitätsäquivalentem Trace ausgeführt, werden die Variablen entsprechend mit den in dessen trace vorkommenden Werten belegt. Es werden in keinen Ablauf zwei Variablen mit dem gleichen Wert belegt.

Definition 13.6 (Identitätshistorybaum) *Sei $A = (Q_A, C, var_A, q_{A,0}, E_A, \Omega_A)$ ein Identitätsautomat. Der* Identitätshistorybaum *zu A der Tiefe $k \in \mathbb{N}$ ist der Serviceautomat $U \underset{Def}{=} (Q, C, var, q_0, E, \Omega)$ mit*

- $Q = \Sigma^k(A)\backslash \sim_{id}$.
- $q_0 = [\varepsilon]_{id}$
- $\Omega = Q$

Die Variablenmenge jedes Knotens q mit dem Repräsentanten w_q sei

$$var(q) = \{x_u | u \in \mathcal{U}, u \text{ kommt in einer Aktion in } w_q \text{ vor}\}$$

Die Kantenmenge E sei folgendermaßen definiert:
$([w]_{id}, c, G_{[w]_{id},[wa]_{id}}, [wa]_{id}) \in E$ gdw. $w \in \Sigma^{k-1}(A)$, $a \in \Sigma(A)$.

Den Guard für eine Kante zwischen zwei Knoten q, q' mit den Repräsentanten w_q und $w_{q'} = w_q\, a'$ definieren wir durch

$$\begin{aligned} G_{q,q'} \underset{Def}{=} & \bigwedge_{x \in var(q)} x^{\text{new}} = x \\ & \wedge \bigwedge_{x,y \in var(q')} x^{\text{new}} \neq y^{\text{new}} \\ & \wedge \bigwedge_{0 \leq i \leq n-1} x_{u_i}^{\text{new}} = c_i \end{aligned}$$

Dabei sei $a' = (c, \gamma)$, $var(c) = \{c_0, \ldots, c_{n-1}\}$ und $u_i = \gamma(c_i)$ für $0 \leq i \leq n-1$.

Die Wahl des Repräsentanten eines Knotens beeinflusst die Namensgebung seiner Variablen, nicht jedoch die Struktur und die Semantik des Identitätshistorybaumes.

Nach Konstruktion des Identitätshistorybaumes sind alle erreichbaren Instanzen eines Knotens zueinander identitätsäquivalent. Jeder Knoten entspricht genau einer Äquivalenzklasse. Der symbolische Erreichbarkeitsgraph des Identitätshistorybaumes ist daher isomorph zu diesem.

Korollar 13.5 *Die erreichbaren Instanzen jedes Knotens des Identitätshistorybaumes sind paarweise zueinander identitätsäquivalent.*

Ebenfalls nach Konstruktion ist das Transitionssystem des Identitätshistorybaumes ein deterministischer Baum: Jedes Eingabewort w legt eindeutig einen Zustand z fest und zu jedem Zustand z gibt es genau ein Eingabewort w, welches zu diesem führt.

Korollar 13.6 *Sei A ein Identitätsautomat. Der Erreichbarkeitsgraph des Identitätshistorybaumes zu A der Tiefe k ist isomorph zum kanonischen Baum zu der Sprache $L = \Sigma^k(A)$ und damit isomorph zur kanonischen Überapproximation (Def. 10.1) zu A der Tiefe k.*

Aufgrund der eins-zu-eins-Beziehung zwischen Knoten des Identitätshistorybaumes und den Knoten des symbolischen Erreichbarkeitsgraphen können einzelne Knoten des symbolischen Erreichbarkeitsgraphen gezielt entfernt werden. Dies nutzen wir bei der Partnersynthese aus.

Für einen gegebenen Identitätsautomaten A synthetisieren wir einen Partner, indem wir Knoten aus dem Identitätshistorybaum entfernen. So, wie wir in Kapitel 10 zwischen *guten* und *schlechten* Zuständen der Überapproximation unterschieden haben, unterscheiden wir zwischen guten und schlechten Knoten. Das Entfernen eines einzelnen schlechten Knotens aus dem Identitätshistorybaum entspricht dem Entfernen einer (im allgemeinen unendlich großen) Menge von schlechten Zuständen. Wir entfernen solange schlechte Knoten aus dem Identitätshistorybaum, bis dieser nur noch gute Knoten hat oder alle Knoten entfernt wurden. Hat der resultierende Identitätsautomat mindestens einen Knoten, ist er ein Partner von A. Andernfalls ist A nicht bedienbar.

Jeder Knoten des Identitätshistorybaum entspricht einer Klasse identitätsäquivalenter Zustände. Identitätsäquivalente Zustände verhalten sich gleich bezüglich der Eigenschaft, gute oder schlechte Zustände zu sein.

Lemma 13.7 (Symmetrie guter Zustände) *Seien A, B interfaceäquivalente Identitätsautomaten und $(q, \beta), (q, \beta')$ identitätsäquivalente Zustände von B. Dann ist (q, β) gut bzgl. A gdw. (q, β') gut bzgl. A.*

Beweis Nach Korollar 13.1 gibt es eine Bijektion $\sigma : \mathcal{U} \to \mathcal{U}$, so dass $\beta' = \beta \circ \sigma$.

Sei (q, β') ein guter Zustand und sei $(p, \alpha) \in \text{know}_{A,B}(q, \beta)$. Nach Def. (Wissensfunktion) ist (p, q, α, β) in $A \times B$ erreichbar. Wegen Symmetrie (Lemma 13.2) ist $(p, q, \alpha \circ \sigma, \beta \circ \sigma)$ in $A \times B$ erreichbar. Nach Voraussetzung ist von $(p, q, \alpha \circ \sigma, \beta \circ \sigma)$ ein Endzustand von $A \times B$ erreichbar. Wegen Symmetrie (Lemma 13.2) ist auch von (p, q, α, β) ein Endzustand erreichbar. Nach Def. (guter Zustand) ist (q, β) ein guter Zustand. □

Diese Symmetrie rechtfertigt die Erweiterung der Definition des guten Zustandes auf Klassen identitätsäquivalenter Zustände.

Definition 13.7 (Gute Identitätsäquivalenzklasse) *Seien A, B Identitätsautomaten, z ein bzgl. A guter Zustand von B. Dann nennen wir $[z]_{id}$* gut *(bzgl. A). Andernfalls nennen wir $[z]_{id}$* schlecht *(bzgl. A).*

Jeder Knoten des Identitätshistorybaumes entspricht nach Konstruktion genau einer Identitätsäquivalenzklasse. Daher bezeichnen wir einen Knoten nach seiner Klasse ebenfalls als gut oder schlecht.

Wir benötigen nun noch ein Verfahren, um zu entscheiden, ob eine Klasse gut oder schlecht ist. Ein direktes Prüfen, ob der Repräsentant der Klasse nach Def. 10.2 ein guter Zustand ist, ist nicht möglich, da das Wissen des Zustandes im allgemeinen eine unendlich große Menge ist. Um eine endliche Darstellung des Wissens eines Zustandes zu erhalten, erweitern wir die Wissensfunktion know auf Klassen identitätsäquivalenter Zustände.

Bei der Erweiterung der Wissensfunktion auf Klassen ist eine subtile Änderung der Definition notwendig, damit aus der symbolischen Wissensfunktion die gewöhnliche Wissensfunktion know durch Entfaltung rekonstruiert werden kann.

Die Wissensfunktion ordnet jedem Zustand eines Serviceautomaten B eine Menge von Zuständen eines Serviceautomaten A zu. Eine direkte Verallgemeinerung der Wissensfunktion würde jeder Klasse von Zuständen von B eine Menge von Klassen von Zuständen von A zuordnen. Bei einer Verallgemeinerung dieser Art ist jedoch nicht eindeutig rekonstruierbar, welchem Zustand einer Klasse von B welche Menge von Zuständen von A zugeordnet wird.

Die hier definierte *symbolische Wissensfunktion* ordnet statt dessen jeder Klasse von Zuständen von B eine Menge von Klassen von Zuständen des *Kreuzproduktes* $A \times B$ zu. Bei Bildung der Äquivalenzklassen auf den Zuständen des Kreuzproduktes von A und B (statt auf den Zutänden von B) bleibt die Zuordnung zwischen den Zuständen von A und B erhalten.

Definition 13.8 (Symbolische Wissensfunktion) *Seien A, B interfaceäquivalente Identitätsautomaten. Das* symbolische Wissen *eines Zustandes (q, β) von B sei definiert als*

$$\text{sknow}_{A,B}(q,\beta) \underset{Def}{=} \{[p,q,\alpha,\beta]_{id} | (p,q,\alpha,\beta) \text{ erreichbar in } A \times B\}$$

Die symbolische Wissensfunktion sknow *erweitern wir kanonisch auf Klassen identitätsäquivalenter Zustände. Für die Äquivalenzklasse von (q, β) definieren wir*

$$\text{sknow}_{A,B}([q,\beta]_{id}) \underset{Def}{=} \text{sknow}(q,\beta)$$

Beispiel Abb. 13.7 zeigt den symbolischen Erreichbarkeitsgraphen von $\mathsf{FailOnEq} \times \mathsf{U}_0$ und Tabelle 13.1 die zugehörige auf symbolische Wissensfunktion.

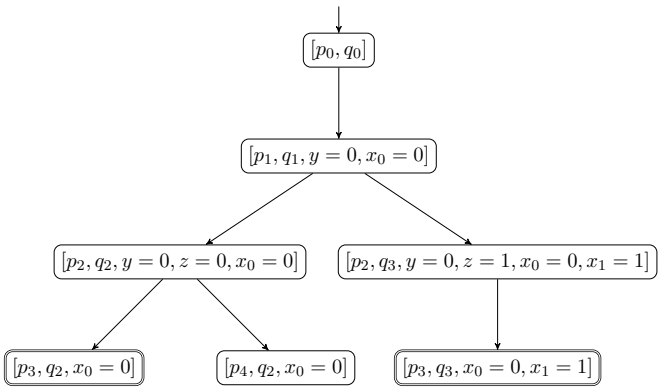


Abbildung 13.7.: Symbolischer Erreichbarkeitsgraph von $\mathsf{FailOnEq} \times \mathsf{U}_0$

Z	$\mathrm{sknow}_{\mathsf{FailOnEq},\mathsf{U}_0}(Z)$
$[q_0]$	$\{[p_0, q_0]\}$
$[q_1, x_0 = 0]$	$\{[p_1, q_1, y = 0, x_0 = 0]\}$
$[q_2, x_0 = 0]$	$\{[p_2, q_2, y = 0, z = 0, x_0 = 0], [p_3, q_2, x_0 = 0], [p_4, q_2, x_0 = 0]\}$
$[q_3, x_0 = 0, x_1 = 1]$	$\{[p_2, q_3, y = 0, z = 1, x_0 = 0, x_1 = 1], [p_3, q_3, x_0 = 0, x_1 = 1]\}$

Tabelle 13.1.: Symbolisches Wissen von U_0 über $\mathsf{FailOnEq}$

Es ist $[p_2, q_3, y = 0, z = 1, x_0 = 0, x_1 = 1]_{id} \in \mathrm{sknow}_{\mathsf{FailOnEq},\mathsf{U}_0}([q_3, x_0 = 0, x_1 = 1]_{id})$.

Es erscheint zunächst überflüssig, dass der Repräsentant $(q_3, x_0 = 0, x_1 = 1)$ der Klasse $[q_3, x_0 = 0, x_1 = 1]_{id})$ auch Bestandteil des Repräsentanten der Klasse $[p_2, q_3, y = 0, z = 1, x_0 = 0, x_1 = 1]_{id}$ ist. Würden wir diesen Bestandteil jedoch weglassen und der Klasse $[q_3, x_0 = 0, x_1 = 1]_{id}$ die Klasse $[p_2, y = 0, z = 1]_{id}$ statt der Klasse $[p_2, q_3, y = 0, z = 1, x_0 = 0, x_1 = 1]_{id}$ zuordnen, könnten wir die erreichbaren Zustände von $A \times B$ nicht mehr korrekt rekonstruieren. Es ist nämlich $[p_2, y = 0, z = 1]_{id} = [p_2, y = 1, z = 0]_{id}$. Setzen wir die Repräsentanten dieser beiden Klassen mit dem Repräsentanten $(q_3, x_0 = 0, x_1 = 1)$ zusammen, erhalten wir die beiden Zustände $(p_2, q_3, y = 0, z = 1, x_0 = 0, x_1 = 1)$ und $(p_2, q_3, y = 1, z = 0, x_0 = 0, x_1 = 1)$. Letzterer ist im Gegensatz zu ersterem in $\mathsf{FailOnEq} \times \mathsf{U}_0$ nicht erreichbar.

Die Erweiterung der symbolischen Wissensfunktion sknow auf Klassen identitätsäquivalenter Zustände ist wohldefiniert.

Korollar 13.8 (Wohldefiniertheit) *Es gilt* $\mathrm{sknow}_{A,B}(z_B) = \mathrm{sknow}_{A,B}(z'_B)$ *für identitätsäquivalente Zustände* z_B *und* z'_B *von* B.

Beweis Nach Def. (symb. Wissen) und wegen Symmetrie (Korollar 13.3) gilt $(p, q, \alpha, \beta) \in \mathrm{sknow}_{A,B}(q, \beta)$ gdw. $(p, q, \alpha \circ \sigma, \beta \circ \sigma) \in \mathrm{sknow}_{A,B}(q, \beta \circ \sigma)$ für jede Bijektion $\sigma : \mathcal{U} \to \mathcal{U}$. Mit Korollar 13.1 folgt die Behauptung. □

Der Bezug zwischen der Wissensfunktion und der symbolischen Wissensfunktion ergibt sich direkt aus ihrer Definition.

Korollar 13.9 *Seien A, B interfaceäquivalente Identitätsautomaten und (q, β) ein Zustand von B. Dann gilt*

$$\begin{aligned} & (p, \alpha) \in \mathrm{know}_{A,B}(q, \beta) \\ \textit{gdw. } & (p, q, \alpha, \beta) \textit{ erreichbar in } A \times B \\ \textit{gdw. } & [p, q, \alpha, \beta]_{id} \in \mathrm{sknow}_{A,B}([q, \beta]_{id}) \end{aligned}$$

Mit der symbolischen Wissensfunktion und dem symbolischen Erreichbarkeitsgraphen wir entscheiden, ob eine Klasse von Zuständen gut oder schlecht ist.

Lemma 13.10 (Gute Identitätsäquivalenzklasse) *Seien A, B interfaceäquivalente Identitätsautomaten, Z eine Klasse identitätsäquivalenter Zustände von $A \times B$. Dann ist Z gut (bzgl. A) genau dann wenn im symbolischen Erreichbarkeitsgraphen von $A \times B$ von jeder Klasse $Z' \in \mathrm{sknow}(Z)$ ein Endknoten erreichbar ist.*

Beispiel Die Klasse $[q_2, x_0 = 0]$ von $\mathsf{U_0}$ ist schlecht bzgl. $\mathsf{FailOnEq}$, da vom Knoten $[p_3, q_2, x_0 = 0] \in \mathrm{sknow}_{\mathsf{FailOnEq},\mathsf{U_0}}([q_2, x_0 = 0])$ kein Endknoten im symbolischen Erreichbarkeitsgraphen von $\mathsf{FailOnEq} \times \mathsf{U_0}$ erreichbar ist. Damit ist q_2 ein schlechter Knoten von $\mathsf{U_0}$.

Algorithmus 3 Partnersynthese für azyklische Identitätsautomaten

Eingabe: Ein azyklischer Identitätsautomat A der Tiefe $k \in \mathbb{N}$
Sei $U_0 = (Q, C, var, q_0, E, \Omega)$ der Identitätshistorybaum zu A der Tiefe k
while es ex. schlechter Knoten q von U_i:
    U_{i+1} := Einschränkung von U_i auf die (bzgl. A) guten Knoten von U_i
if U_i enthält q_0
    return U_i
else
    return „A ist nicht bedienbar“

Nun können wir den Algorithmus zur Synthese eines Partners eines azyklischen Identitätsautomaten formulieren. Genau so, wie der Algorithmus 2 aus Kapitel 10 schlechte Zustände aus dem kanonischen Baum entfernt, entfernt Algorithmus 3 iterativ die schlechten Knoten aus dem Identitätshistorybaum.

Beispiel Durch Entfernen von q_2 entsteht $\mathsf{U_1}$ (Abb. 13.8) aus $\mathsf{U_0}$. Abb. 13.9 zeigt den symbolischen Erreichbarkeitsgraphen von $\mathsf{U_1}$.

Im symbolischen Erreichbarkeitsgraphen von $\mathsf{FailOnEq} \times \mathsf{U_1}$ gibt es von jedem Knoten einen Pfad zu einem Endknoten. Damit ist $\mathsf{U_1}$ ein Partner von $\mathsf{FailOnEq}$.

Die Korrektheit des Algorithmus 3 folgt direkt aus der Äquivalenz des Identitätshistorybaumes zum kanonischen Baum und der Äquivalenz schlechter Zustände und schlechter Knoten des Identitätshistorybaumes.

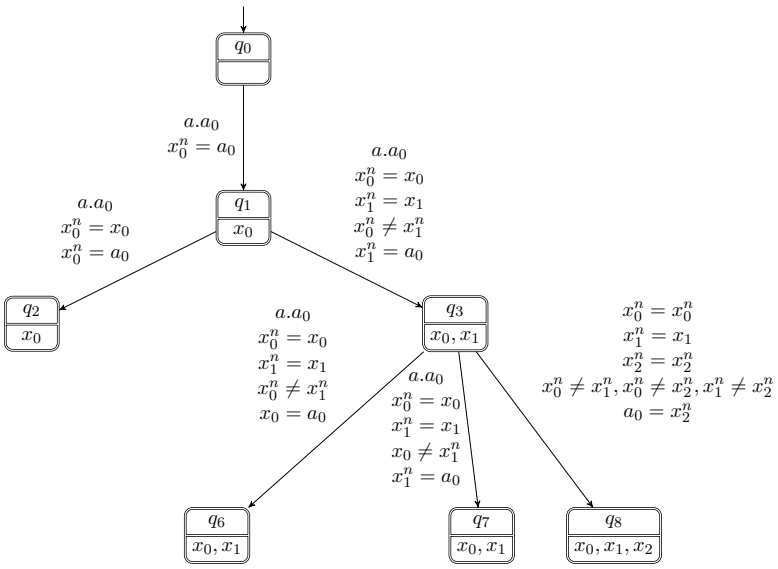


Abbildung 13.8.: Serviceautomat $\mathsf{U_1}$

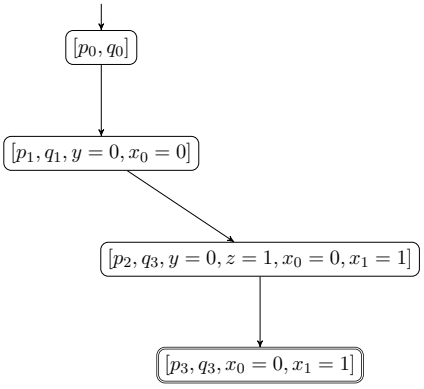


Abbildung 13.9.: Symbolischer Erreichbarkeitsgraph von $\mathsf{FailOnEq} \times \mathsf{U_1}$

13.3. Verschmelzen von Knoten

In diesem Kapitel stellen wir eine Regel zum Verschmelzen von Knoten eines Partners eines Identitätsautomaten vor. Wir zeigen außerdem, wie redundante Variablen eines Partners eines Identitätsautomaten berechnet werden können.

In Abschnitt 5.3 haben wir gezeigt, dass Zustände eines Partners eines azyklischen Serviceautomaten, welche das gleiche Wissen über den Serviceautomaten haben, verschmolzen werden können. Der Partner behält dabei seine Eigenschaft, ein Partner des gegebenen Serviceautomaten zu sein. Dieses Resultat für einzelne Zustände verallgemeinern wir auf Knoten eines Identitätsautomaten.

Wir zeigen, dass aus einem Partner eines Identitätsautomaten durch Verschmelzen zweier Knoten mit äquivalentem symbolischen Wissen wieder ein Partner hervorgeht.

Die beiden Knoten, die verschmolzen werden, müssen zudem zwei technische Nebenbedingungen erfüllen:

- Die beiden Knoten haben die gleiche Variablenmenge
- Die erreichbaren Instanzen jedes Knotens sind jeweils zueinander identitätsäquivalent

Die Gleichheit der Variablenmengen der beiden Knoten wird in der Definition der in Abschnitt 2.4.2 eingeführten Verschmelzungsoperation (Def. 2.29) vorausgesetzt und ist notwendig, damit dem durch Verschmelzen entstehenden Knoten wieder eindeutig eine Menge von Variablen zugeordnet werden kann. Gegebenenfalls können Variablen vor dem Verschmelzen geeignet umbenannt werden. In jedem Fall können nur Knoten mit der gleichen Anzahl Variablen verschmolzen werden.

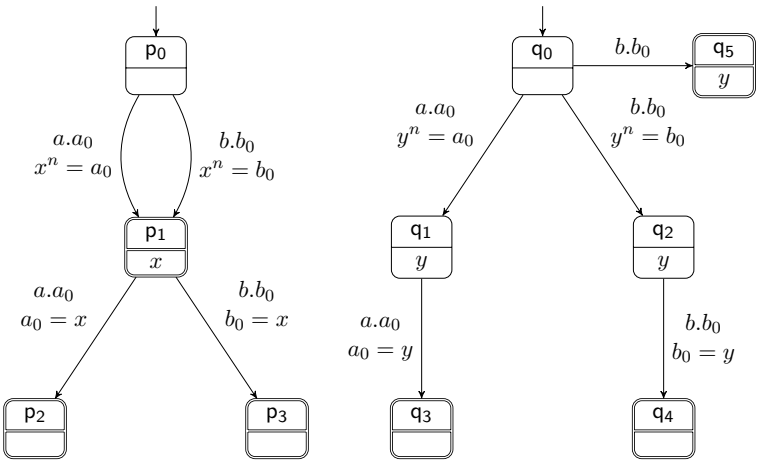
Die Äquivalenz der erreichbaren Instanzen der Knoten ist notwendig, damit beiden Knoten des Partners jeweils genau ein Knoten des symbolischen Erreichbarkeitsgraphen zugeordnet werden kann. Falls sich die erreichbaren Instanzen eines Knotens auf mehrere Äquivalenzklassen verteilen, kann der Identitätsautomat gegebenenfalls durch Aufspalten von Knoten verfeinert werden, so dass diese Eigenschaft erfüllt ist.

Knoten mit gleichem symbolischen Wissen nennen wir *symbolisch wissensäquivalent*.

Zum Vergleichen des symbolischen Wissens zweier Knoten q_1 und q_2 benötigen wir aus technischen Gründen eine Operation zum Umbenennen von Knoten. Das symbolische Wissen eines Knotens q eines Partners B eines Identitätsautomaten A besteht aus Identitätsäquivalenzklassen der Zustände des Kreuzproduktes von $A \times B$. Der Knoten q ist daher selbst Bestandteil der Repräsentanten der ihm zugeordneten der Identitätsäquivalenzklassen.

Beim Vergleichen des symbolischen Wissens zweier Knoten q_1 und q_2 von B wollen wir daher die Knoten q_1 und q_2 ignorieren und die Repräsentanten der Identitätsäquivalenzklassen nur anhand der Variablenbelegung und des Knotens von A vergleichen. Der technisch einfachste Weg, Repräsentanten auf diese Weise zu vergleichen, besteht darin, die Knoten q_1 und q_2 in den Repräsentanten umzubenennen.

Zu diesem Zweck definieren wir die Umbenennungsoperation RenameNode , die einen Knoten in einem Zustand des Kreuzproduktes ersetzt. Für Knoten q_1, q_2 von B und



(a) C (b) D

Abbildung 13.10.: Zwei Identitätsautomaten, die Partner sind

einen Zustand (p, α, q_1, β) von $A \times B$ sei

$$\text{RenameNode}_{q_1 \to q_2}(p, \alpha, q_1, \beta) \underset{Def}{=} (p, \alpha, q_2, \beta)$$

Die Umbenenungsoperation wenden wir kanonisch auf Klassen von Zuständen und Mengen von Klassen an. Das symbolische Wissen zweier Knoten q_1 und q_2 wird dadurch vergleichbar. Zwei Knoten q_1 und q_2 von B sind *symbolisch wissensäquivalent*, wenn sie das gleiche symbolische Wissen haben unter der Voraussetzung, dass q_1 und q_2 in den verglichenen Zuständen nicht unterschieden werden.

Definition 13.9 (Symbolisch wissensäquivalente Knoten) *Seien A und B interfaceäquivalente Identitätsautomaten, q_1, q_2 Knoten von B mit jeweils gleicher Variablenmenge und Z_1 bzw. Z_2 die Menge der erreichbaren Instanzen von q_1 bzw. q_2. Die erreichbaren Instanzen seien jeweils zueinander identitätsäquivalent, d. h. Z_1 und Z_2 seien Klassen identitätsäquivalenter Zustände.*

Sei $\text{sknow}_{A,B}(Z_1) = \text{RenameNode}_{q_1 \to q_2}(\text{sknow}_{A,B}(Z_2))$. *Dann nennen wir q_1 und q_2* symbolisch wissensäquivalent.

Beispiel Abb. 13.10 zeigt zwei Identitätsautomaten C und D. Der Serviceautomat D ist isomorph zu einem symbolischen Erreichbarkeitsgraphen. Die Klassen $[q_1, y = 0]_{id}$ und $[q_2, y = 0]_{id}$ enthalten jeweils genau die in D erreichbaren Instanzen von q_1 bzw. q_2.

Die Knoten q_1 und q_2 sind symbolisch wissensäquivalent, denn es gilt

$$\text{RenameNode}_{q_1 \to q_2}([p_1, q_1, x = 0, y = 0]_{id}) = [p_1, q_2, x = 0, y = 0]_{id}$$

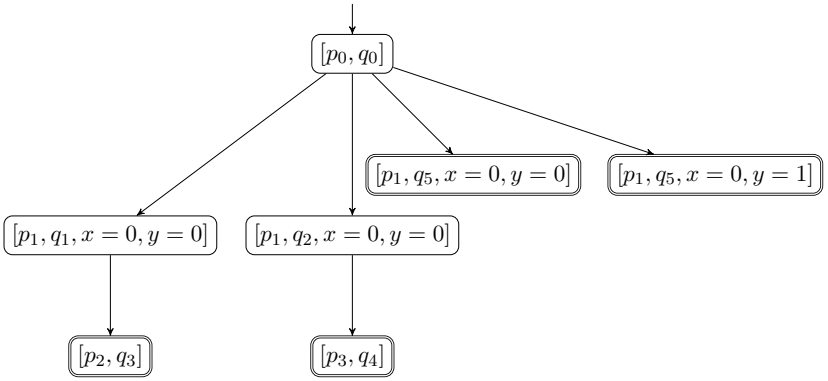


Abbildung 13.11.: Symbolischer Erreichbarkeitsgraph von $\mathsf{C} \times \mathsf{D}$

Z	$\text{sknow}_{\mathsf{C},\mathsf{D}}(Z)$
$[q_0]$	$\{[p_0, q_0]\}$
$[q_1, y = 0]$	$\{[p_1, q_1, x = 0, y = 0]\}$
$[q_2, y = 0]$	$\{[p_1, q_2, x = 0, y = 0]\}$
$[q_3]$	$\{[p_2, q_3]\}$
$[q_4]$	$\{[p_3, q_4]\}$
$[q_5, y = 0]$	$\{[p_1, q_5, x = 0, y = 0], [p_1, q_5, x = 0, y = 1]\}$

Tabelle 13.2.: Symbolisches Wissen von D über C

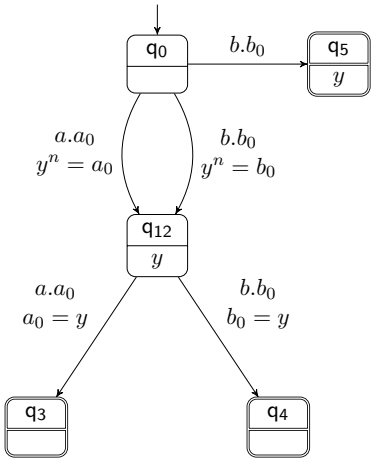


Abbildung 13.12.: D nach Verschmelzen von q_1 und q_2

d. h., die Repräsentanten der mit den Knoten assoziierten Klassen (und damit die Klassen selbst) gehen durch Umbenennen von q_1 in q_2 und umgekehrt auseinander hervor. Keine weiteren Knoten von D sind symbolisch wissensäquivalent.

Der Serviceautomat in Abb. 13.12, der aus D durch Verschmelzen von q_1 und q_2 entsteht, ist wieder ein Partner von C.

Die Äquivalenz des symbolischen Wissens zweier Knoten q_1 und q_2 impliziert die Gleichheit des Wissens von Paaren von Instanzen der beiden Knoten, die wir bei gleicher Belegung der Variablen der beiden Knoten erhalten.

Lemma 13.11 *Seien A und B interfaceäquivalente Identitätsautomaten und seien q_1, q_2 symbolisch wissensäquivalente Knoten von B.*

Dann gilt $\text{know}_{A,B}([q_1, \beta]_{id}) = \text{know}_{A,B}([q_2, \beta]_{id})$ *für jede Belegung β der Variablen von q_1 bzw. q_2.*

Beweis Sei $(q_A, \alpha) \in \text{know}_{A,B}([q_1, \beta]_{id})$. Nach Def. (Wissen) ist (p, q_1, α, β) in $A \times B$ erreichbar. Nach Def. (symb. Wissen) ist $[p, q_1, \alpha, \beta]_{id} \in \text{sknow}([z_1]_{id})$ für jede in B erreichbare Instanz z_1 von q_1. Nach Voraussetzung ist auch $[p, q_2, \alpha, \beta]_{id} \in \text{sknow}([z_2]_{id})$ für jede in B erreichbare Instanz z_2 von q_2. Nach Def. (symb. Wissen) ist (p, q_2, α, β) erreichbar in $A \times B$. Nach Def. (Wissen) ist $(p, \alpha) \in \text{know}_{A,B}([q_2, \beta]_{id})$. □

Aufgrund der Äquivalenz des Wissens und des symbolischen Wissens können wir das in Abschnitt 5.3 für einzelne Zustände gezeigte Theorem 5.5 auf Knoten eines Identitätsautomaten verallgemeinern. Das Verschmelzen symbolisch wissensäquivalenter Knoten eines azyklischen Partners erhält die Partnereigenschaft.

Theorem 13.12 *Seien A und B azyklische Identitätsautomaten, die Partner sind und seien q_1, q_2 symbolisch wissensäquivalente Knoten von B, so dass keiner der beiden Knoten ein Nachfahre des jeweils anderen ist.*

Sei B' der Serviceautomat, der aus B durch Verschmelzen (nach Def. 2.29) von q_1 und q_2 hervorgeht. Dann ist B' ein Partner von A.

Beweis Semantisch entspricht das Verschmelzen von q_1 und q_2 dem Verschmelzen der Zustände (q_1, β) und (q_2, β) für jede Belegung β (Korollar 2.9). Da (q_1, β) und (q_2, β) nach Lemma 13.11 das gleiche Wissen über A haben, bleibt nach Theorem 5.5 die Partnereigenschaft erhalten. □

Dieses Theorem ist insbesondere in Hinblick auf die Synthese zyklischer Partner interessant. Die Synthese eines Partners für einen zyklischen Identitätautomaten ist ein offenes Problem. Wie in Abschnitt 6.2 gezeigt, gibt es innerhalb der Klasse der Identitätsautomaten nicht immer einen endlich darstellbaren Partner.

In den Fällen, in denen ein endlich darstellbarer zyklischer Partner existiert, könnte ein solcher möglicherweise durch Faltung aus einem nicht tiefenbeschränkten Identitätshistorybaum erzeugt werden. Unter günstigen Umständen könnte dieser zu einem zyklischen Serviceautomaten mit endlich vielen Knoten zusammengefaltet werden. Wir wollen diese Idee hier kurz skizzieren.

Bei der Faltung werden alle Knoten mit äquivalentem symbolischen Wissen zu jeweils einem Knoten verschmolzen. Vor dem Falten ist es notwendig, so viele überflüssige Variablen des Identitätshistorybaumes zu eliminieren wie möglich. Andernfalls hätte die symbolische Wissensäquivalenz stets unendlich viele Klassen: Die Anzahl der Variablen der Knoten des Identitätshistorybaumes wächst unbeschränkt mit dessen Tiefe. Knoten mit einer unterschiedlichen Anzahl von Variablen sind grundsätzlich nie symbolisch wissensäquivalent.

Möglicherweise existiert eine Teilklasse von Identitätsautomaten, die jeweils mindestens einen speicherbeschränkten Partner haben, falls sie bedienbar sind. Für eine solche Klasse von Identitätsautomaten hätte der Identitätshistorybaum nach dem Eliminieren aller redundanten Variablen möglicherweise endlich viele symbolisch wissensäquivalente Knoten. Aus dem gefalteten Identitätshistorybaum könnte anschließend durch Entfernen von Knoten ein Partner erzeugt werden.

Ob eine Variable x eines Knotens redundant ist, prüft man an der Auswirkung, die ihre Änderung auf das symbolische Wissen der Identitätsäquivalenzklasse des Knotens hat. Ist die Variable x redundant, ändert sich in den Repräsentanten der Klassen, die zusammen das symbolische Wissen des Knotens bilden, jeweils die nur die Belegung von x. Die anderen Bestandteile der Repräsentanten bleiben unverändert.

Lemma 13.13 *Seien A und B interfaceäquivalente Identitätsautomaten und x eine Variable eines Knotens q von B. Dann ist x redundant in q bzgl. A gdw. für jedes Paar (q, β) und (q, β') von erreichbaren Instanzen von q, die sich nur in der Belegung von x unterscheiden und für jeden Zustand (p, α) von A gilt:*

$$[p, q, \alpha, \beta]_{id} \in \text{sknow}_{A,B}([q, \beta]_{id}) \text{ gdw. } [p, q, \alpha, \beta']_{id} \in \text{sknow}_{A,B}([q, \beta']_{id})$$

Beweis Die Behauptung folgt aus der Definition der Redundanz und der Äquivalenz von gewöhnlichem Wissen und symbolischem Wissen. Es ist $\text{know}_{A,B}(q,\beta) = \text{know}_{A,B}(q,\beta')$ äquivalent zu

$$(p,\alpha) \in \text{know}_{A,B}(q,\beta) \text{ gdw. } (p,\alpha) \in \text{know}_{A,B}(q,\beta')$$

Nach Korollar 13.9 ist dies äquivalent zu

$$[p,q,\alpha,\beta]_{id} \in \text{sknow}_{A,B}([q,\beta]_{id}) \text{ gdw. } [p,q,\alpha,\beta']_{id} \in \text{sknow}_{A,B}([q,\beta']_{id})$$

Zusammen mit der Definition der Redundanz folgt die Behauptung. □

Eine entscheidende Aufgabe bei der Lösung des Syntheseproblems für zyklische Identitätsautomaten wird sein, ein hinreichendes Kriterium dafür zu finden, dass die Anzahl der Variablen pro Knoten des Identitätshistorybaumes auf endlich viele reduziert werden kann. Dies ist nicht bei jedem Identitätsautomaten möglich. Wie in Abschnitt 6.2 gezeigt wurde, benötigen Partner mancher Identitätsautomaten im allgemeinen eine unbeschränkt wachsende Anzahl von Variablen.

14. Stand der Technik

Es existieren zahlreiche Arbeiten zur Synthese von Controllern, die auf der Basis von endlichen Automaten modelliert werden [76]. Dennoch gibt es vergleichsweise wenige Arbeiten, die sich explizit mit Daten befassen.

Die frühesten uns bekannten Arbeiten zur Synthese von Controllern für Systeme mit unendlich vielen Zuständen stammen von Kumar und Garg [25, 46, 47]. Kumar et al. stellen ein allgemeines Framework zur Beschreibung eines Systems mit unendlich vielen Zuständen vor. Ein Zustand ist eine Belegung von endlich vielen Variablen mit natürlichen Zahlen. In ihrem Framework ist Bedienbarkeit im allgemeinen unentscheidbar. Es existiert jedoch stets ein eindeutig bestimmter größter Controller. Die Autoren zeigen, dass eine spezielle Klasse von Petrinetzen in ihrem Framework beschrieben werden kann, für die Bedienbarkeit entscheidbar ist. Die Anzahl der Marken auf einem Platz des Petrinetzes korrespondiert dabei jeweils mit der Belegung einer Variable des Frameworks. Das Petrinetz selbst ist ungefärbt.

Kaylon et al. [41, 40] stellen ein auf abstrakter Interpretation [17] basierendes Verfahren zur Synthese eines Controllers für ein offenes System mit unendlich vielen Zuständen vor. Ein Zustand des Systems entspricht einer Belegung einer endlichen Menge globaler Variablen. Ihr Algorithmus garantiert nicht, dass stets ein Controller erzeugt wird, falls ein solcher existiert und kann daher nicht zum Entscheiden der Bedienbarkeit des Systems verwendet werden. Mit Hilfe abstrakter Interpretation können Zustandsmengen mit speziellen Eigenschaften (z. B. Deadlocks) nur näherungsweise berechnet werden. Der Algorithmus entfernt daher oft mehr Zustände als notwendig und erzeugt nicht immer einen größtmöglichen Controller.

Lohmann et al. [58] skizzieren beispielhaft die Synthese eines Partners für ein spezielles algebraisches Petrinetz. Sie geben jedoch keinen allgemeinen Algorithmus zur Partnersynthese. Sie verallgemeinern das in [96] vorgestellte Konstruktionsverfahren für Partner und benutzen parametrisierte Markierungen [81] zum Repräsentieren von Mengen von Zuständen. Die Berechnung der parametrisierter Markierungen basiert auf der Unifikation von Termen. In der ursprünglichen Version des von den parametrisierten Markierungen gebildeten parametrisierten Erreichbarkeitsgraphen kann die Ungleichheit von Termen nicht explizit ausgedrückt werden. Um die Ungleichheit von Termen darstellen zu können, weichen die Autoren vom ursprünglichen Konstruktionsalgorithmus des parametrisierten Erreichbarkeitsgraphen ab und unifizieren bestimmte Terme in einigen symbolischen Markierungen nicht. Statt dessen wird für diese Terme eine Fallunterscheidung vorgenommen und für jede Relation, in denen die Terme stehen können, ein alternativer Zweig im Graphen eingeführt. Diese Relationen werden bei der Formulierung der Guards des Partners berücksichtigt.

Mit dem in [58] vorgeschlagenen Verfahren ist es schwierig, einen Service wie Ran-

domSeq aus Abschnitt 6.2 zu analysieren, da aufgrund des dort beschriebenen Zuordnungsproblems zwischen einer Variable des Service und einer Variable des Partners im allgemeinen keine eindeutigen Relationen zwischen Variablen angegebenen werden können. Der parametrisierte Erreichbarkeitsgraph kann zwar die erreichbaren Zustände eines einzelnen Service darstellen. Er erhält jedoch nicht die Beziehungen zwischen Zuständen zweier miteinander interagierender Services.

Ein termbasierter Ansatz zur Repräsentation von Mengen von Zuständen ähnlich dem des parametrisierten Erreichbarkeitsgraphen könnte jedoch möglicherweise bei der Synthese von Partnern von Services mit speziellen Einschränkungen zum Erfolg führen. Mögliche Kandidaten wären beispielsweise Services ohne Negation, die selbst keine aus einer unendlich großen Domäne nichtdeterministisch gewählten Werte erzeugen. Die Beziehungen zwischen Zuständen zweier miteinander interagierender Services sind unter dieser Voraussetzung etwas einfacher und möglicherweise durch einfachere Beziehungen zwischen Termen ausdrückbar.

Viele Arbeiten zur Controllersynthese, die sich mit Daten beschäftigen, beschränken sich auf Systeme mit endlich vielen Zuständen. Der Zustandsraum der Systeme wird häufig entweder mit Hilfe von BDDs [13] oder explizit durch einen endlichen Automaten dargestellt [69, 84, 85].

Markovsky [60] stellt ein prozessalgebrabasiertes Framework zur Modellierung kommunizierender Systeme vor, welche Daten verarbeiten und untereinander kommunizieren können. Er stellt ein Verfahren zur Synthese eines Controllers eines solchen Systems vor. Für das kontrollierte System können verschiedene Anforderungen spezifiziert werden, die erfüllt werden sollen, darunter auch Deadlockfreiheit. Der Ansatz setzt voraus, dass neben dem zu kontrollierenden System ein weiteres *Observer* genanntes System gegeben ist, welches die Ereignisse des kontrollierten Systems beobachtet und auf dessen Variablen der Controller lesend zugreifen kann.

Zur Controllersynthese für ungefärbte Petrinetze gibt es zahlreiche Arbeiten (siehe [36] für einen Überblick). Für gefärbte Petrinetze gibt es mehrere Algorithmen zur Synthese von Controllern, welche auf den Well-Formed Nets (WFN) [15] und ihrem symbolischen Erreichbarkeitsgraphen aufbauen.

Abid et al. [2] stellen drei verschiedene Ansätze zur Synthese eines Controllers für ein WFN vor. Der synthetisierte Controller des ersten Ansatzes besteht aus einem einzigen Platz, der über Kanten mit Transitionen des WFN verbunden ist. Die Bögen werden mit Hilfe der *Regionentheorie* [5] berechnet. Der Controller sorgt dafür, dass das WFN deadlockfrei ist und keine *verbotenen Markierungen* erreicht werden. Der zweite Ansatz optimiert den Synthesealgorithmus, indem die Berechnung der Regionen auf dem symbolischen Erreichbarkeitsgraphen ausgeführt wird. Der dritte Ansatz stellt einen Controller mit mehreren Plätzen vor.

Thierry-Mieg et al. [90] kombinieren den symbolischen Erreichbarkeitsgraphen eines WFN mit *Data Decision Diagrams* [18] zur Darstellung des Erreichbarkeitsgraphen eines WFN. Dadurch erhalten sie eine noch kompaktere Darstellung des Erreichbarkeitsgraphen.

15. Zusammenfassung

In diesem Teil haben wir die Synthese eines Partners für zwei Klassen von Services mit unendlich vielen Zuständen gezeigt. Ein Synthesealgorithmus für Partner dient einerseits dazu, die Bedienbarkeit von Services konstruktiv zu entscheiden. Eine weitere Anwendung besteht in der Synthese von Adaptern.

Wir haben zunächst einen Algorithmus zum Synthetisieren von Partnern azyklischer Serviceautomaten mit endlich vielen Zuständen vorgestellt und dessen Korrektheit bewiesen. Der Algorithmus setzt formal synchrone Kommunikation zwischen den Serviceautomaten voraus.

Das Syntheseproblem für asynchron kommunizierende Serviceautomaten kann auf das Syntheseproblem für synchron kommunizierende Serviceautomaten zurückgeführt werden. Da azyklische Services nur endlich viele Nachrichten miteinander austauschen können, ist die Anzahl der Nachrichten in jedem Nachrichtenpuffer beschränkt. Im Gegensatz zu dem in [96] beschriebenen Synthesealgorithmus für asynchron kommunizierende zyklische Serviceautomaten muss im azyklischen Fall die Maximalzahl der Nachrichten in einem Puffer nicht explizit beschränkt werden, um einen endlichen Zustandsraum zu garantieren.

Den Synthesealgorithmus für Serviceautomaten mit endlich vielen Zuständen haben wir auf azyklische Serviceautomaten mit unendlich vielen Zuständen verallgemeinert. Wir haben *Formelautomaten*, eine Klasse von Serviceautomaten mit unendlich vielen Zuständen definiert, deren Guards durch Formeln der Prädikatenlogik erster Stufe dargestellt werden. Ein Formelautomat spezifiziert die Verarbeitung von Daten durch Funktionen, Relationen und Variablen.

Für einen Formelautomaten haben wir gezeigt, wie Mengen von Zuständen mit bestimmten Eigenschaften mit Hilfe prädikatenlogischer Formeln beschrieben werden können. Für eine geeignet gewählte Interpretation ist die Erfüllbarkeit der Formeln entscheidbar. Für diese Interpretation ist z. B. entscheidbar, ob ein Serviceautomat schwach terminiert.

Eine prädikatenlogische Formel erlaubt eine feinere Beschreibung einer Zustandsmenge, als z. B. eine intervallbasierte Darstellung von Zuständen. Intervallbasierte Abstraktionen von Zuständen werden häufig im Zusammenhang mit abstrakter Interpretation [17] verwendet. Mit prädikatenlogischen Formeln können wir insbesondere Mengen von Zuständen eines Serviceautomaten beschreiben, deren Eigenschaften durch Beziehungen zu Zuständen eines anderen Serviceautomaten definiert werden. Wie wir in Teil 2 gesehen haben, können die Beziehungen von Zuständen zweier Identitätsautomaten zueinander nicht durch Intervalle dargestellt werden.

Die Bedienbarkeit eines Formelautomaten ist für eine geeignet gewählte Interpretation ebenfalls entscheidbar. Wir stellen einen Algorithmus zum Synthetisieren eines Partners

eines azyklischen Serviceautomaten vor. Die Schritte, die dieser Algorithmus ausführt, sind jeweils äquivalent zu den Schritten, die der entsprechende Algorithmus für Serviceautomaten mit endlich vielen Zuständen ausführt. Jede Operation des Algorithmus auf dem unendlich großen Transitionssystem eines Serviceautomaten beschreiben wir durch eine Operation auf den Guards des Serviceautomaten. Mit Hilfe dieses Algorithmus kann die Bedienbarkeit spezieller Formelautomaten wie z. B. Presburgerautomaten entschieden werden.

Wir haben außerdem demonstriert, wie der Synthesealgorithmus für Partner von Formelautomaten zur Synthese von Adaptern eingesetzt werden kann. Um praxistaugliche Adapter zu synthetisieren, muss der Ansatz um die Möglichkeit erweitert werden, Nebenbedingungen zu spezifizieren, die der Adapter einhalten muss. Diese Erweiterung bleibt Gegenstand zukünftiger Arbeiten.

Offen bleibt auch die Frage, unter welchen Bedingungen es möglich ist, einen Partner eines zyklischen Serviceautomaten zu synthetisieren. Im allgemeinen ist die Existenz eines Partners eines zyklischen Formelautomaten unentscheidbar. Wir vermuten jedoch, dass eine Teilklasse zyklischer Identitätsautomaten existiert, in der die Bedienbarkeit eines Serviceautomaten entscheidbar ist. Nicht jeder bedienbare Identitätsautomat hat, wie in Teil 2 gezeigt, einen endlich darstellbaren Partner. Ein erster Schritt zur Lösung des Syntheseproblems wäre daher, zunächst eine Klasse von Identitätsautomaten zu finden, in der jeder bedienbare Serviceautomat einen Partner mit endlich vielen Knoten hat.

Die Notwendigkeit, eine spezielle Klasse von Identitätsautomaten zu finden und die endliche Darstellbarkeit von Partnern innerhalb dieser Klasse formal beweisen zu können, motiviert die Einführung eines weiteren Ansatzes zur Synthese von Partnern speziell für Identitätsautomaten. Dieser Ansatz nutzt Symmetriebeziehungen zwischen Abläufen aus, um unendlich große Mengen von Zuständen mit bestimmten Eigenschaften darzustellen. Mengen von Zuständen eines Identitätsautomaten können wir durch Äquivalenzklassen von Zuständen darstellen. Wir haben einen Synthesealgorithmus für azyklische Identitätsautomaten vorgestellt, der auf der äquivalenzklassenbasierten Darstellung von Zuständen aufbaut.

Der äquivalenzklassenbasierte Ansatz bildet eine gute Grundlage für eine Verallgemeinerung des Synthesealgorithmus auf zyklische Identitätsautomaten. Die Anzahl der Knoten, die ein Partner mindestens benötigt, kann durch Äquivalenzklassen einfacher analysiert werden als durch eine allgemeine formelbasierte Darstellung von Zustandsmengen.

Wir definieren eine Äquivalenzrelation zwischen Knoten eines Partners, die *symbolische Wissensäquivalenz*. Die Anzahl der Äquivalenzklassen der symbolischen Wissensäquivalenz hängt maßgeblich von der Anzahl der Variablen des Partners ab. Für einen Partner, dessen Anzahl von Variablen pro Knoten beschränkt ist, ist auch die Anzahl der Klassen der symbolischen Wissensäquivalenz endlich.

Symbolisch wissensäquivalente Knoten eines azyklischen Partners eines Identitätsautomaten können miteinander verschmolzen werden. Hierbei bleibt die Partnerbeziehung der beiden Serviceautomaten erhalten.

Wir vermuten, dass in ähnlicher Weise durch Verschmelzen symbolisch wissensäquiva-

lenter Knoten eine zyklische Variante des Identitätshistorybaumes erzeugt werden kann, aus dem anschließend mit dem in Kapitel 13 vorgestellten Synthesealgorithmus ein ebenfalls zyklischer Partner erzeugt werden kann.

Wir vermuten, dass bei der Konstruktion der zyklischen Variante des Identitätshistorybaumes die Eliminierung redundanter Variablen eine wichtige Rolle spielen würde: Es müssen ausreichend viele Variablen eliminiert werden können, so dass die Knoten des Identitätshistorybaumes sich auf höchstens endlich viele Klassen der symbolischen Wissensäquivalenz verteilen.

Die Ausarbeitung eines Ansatzes zur Synthese von Partnern zyklischer Serviceautomaten bleibt jedoch weiterhin Gegenstand zukünftiger Arbeiten. In dieser Arbeit konnte keine Klasse zyklischer Identitätsautomaten gefunden werden, so dass jeder bedienbare Serviceautomat dieser Klasse garantiert einen endlich darstellbaren Partner innerhalb dieser Klasse hat.

Teil IV.

Zusammenfassung

16. Zusammenfassung und Ergebnisse der Arbeit

Mit diesem Kapitel schließen wir unsere Arbeit ab und fassen den Inhalt und die Ergebnisse der Arbeit zusammen. Wir zeigen weiterhin offene Probleme auf.

Den Ausgangspunkt dieser Arbeit bilden verteilte Systeme, die nach dem Paradigma des Service-Oriented Computing (SOC) aufgebaut sind. Ein solches verteiltes System besteht aus mehreren Services. Die Services interagieren miteinander durch den Austausch von Nachrichten. Die Nachrichten enthalten im allgemeinen Daten. Jeder Service kann Daten speichern und verarbeiten.

In dieser Arbeit haben wir untersucht, wann Services zusammenarbeiten können. Zu diesem Zweck haben wir *Serviceautomaten*, ein formales Modell zur Darstellung von Services definiert. Dieses Servicemodell spezifiziert durch Prädikate, wie Daten durch den Service behandelt werden.

Zwei Services, die miteinander zusammenarbeiten können, nennen wir formal *Partner*. Zwei Serviceautomaten sind *Partner*, wenn sie ein bestimmtes Kompatibilitätskriterium erfüllen. Wir haben in dieser Arbeit das Kompatibilitätskriterium *schwache Terminierung* betrachtet.

Einen Serviceautomaten, der mindestens einen Partner hat, nennen wir *bedienbar*. Die Bedienbarkeit eines Serviceautomaten ist eine grundlegende Voraussetzung dafür, dass der Serviceautomat in einem verteilten System eingesetzt werden kann. In der Arbeit haben wir untersucht, wann ein Serviceautomat bedienbar ist und welche Eigenschaften seine Partner haben.

Für zwei Klassen von Serviceautomaten, *Formelautomaten* und *Identitätsautomaten*, haben wir Algorithmen zum Entscheiden der Bedienbarkeit vorgestellt. Die Serviceautomaten beider Klassen haben unendlich viele Zustände. Die Klassen unterscheiden sich bezüglich der Prädikate, die als Guards eines Serviceautomaten verwendet werden.

Bedienbarkeit von Formelautomaten Ein *Formelautomat* ist ein Serviceautomat, dessen Guards Formeln des Prädikatenkalküls erster Stufe sind. Ein *Presburgerautomat* ist ein Spezialfall eines Formelautomaten, dessen Formeln durch in Presburger-Arithmetik spezifiziert werden. Die Erfüllbarkeit einer Formel in Presburger-Arithmetik ist entscheidbar.

Wir haben in der Arbeit einen Algorithmus zum Synthetisieren eines Partners eines azyklischen Formelautomaten vorgestellt. Mit diesem Algorithmus kann die Bedienbarkeit eines Presburgerautomaten konstruktiv entschieden werden.

Die Ergebnisse zur Bedienbarkeit von Presburgerautomaten fassen wir kurz zusammen:

- Für azyklische Presburgerautomaten ist Bedienbarkeit entscheidbar.
- Für zyklische Presburgerautomaten ist Bedienbarkeit unentscheidbar.

Bedienbarkeit von Identitätsautomaten und Speicherplatzbedarf von Partnern *Identitätsautomaten* sind ein Spezialfall von Presburgerautomaten. Ein Identitätsautomat erlaubt nur Guards, welche die paarweise Gleichheit bzw. Ungleichheit von Variablen ausdrücken können. Identitätsautomaten sind die bezüglich der Ausdrucksfähigkeit ihrer Guards einfachste Klasse von Serviceautomaten mit unendlich vielen Zuständen, innerhalb der sinnvolle Aussagen über die Bedienbarkeit eines Serviceautomaten gemacht werden können.

Identitätsautomaten eigenen sich besonders gut dazu, zu untersuchen, wie groß ein Serviceautomat sein muss und wie viel Speicherplatz er mindestens benötigt, um mit einem anderen Serviceautomaten zusammenarbeiten zu können.

Wir haben einen Algorithmus zur Synthese eines Partners eines azyklischen Serviceautomaten vorgestellt. Dieser benutzt Äquivalenzklassen zur Repräsentation unendlich großer Mengen von Zuständen. Mit Äquivalenzklassen kann die Anzahl der Knoten, die zur Darstellung eines Partners notwendig sind, einfacher abgeschätzt werden, als dies mit einer formelbasierten Repräsentation von Zustandsmengen möglich wäre. Für eine zukünftige Verallgemeinerung des Synthesealgorithmus auf zyklische Serviceautomaten könnte der äquivalenzklassenbasierte Ansatz Vorteile bieten.

Die beiden wichtigsten Resultate zu Identitätsautomaten fassen wir kurz zusammen:

- Es existiert ein bedienbarer zyklischer Identitätsautomat, der keinen endlich darstellbaren Partner in der Klasse der Identitätsautomaten hat.
- Für azyklische Identitätsautomaten ist Bedienbarkeit entscheidbar und jeder bedienbare Identitätsautomat hat einen endlich darstellbaren Partner.

Die Entscheidbarkeit der Bedienbarkeit eines Identitätsautomaten folgt im azyklischen Fall trivial aus der Entscheidbarkeit der Bedienbarkeit für Presburgerautomaten. Um die Bedienbarkeit eines azyklischen Identitätsautomaten zu entscheiden, ist der Synthesealgorithmus für Identitätsautomaten also nicht notwendig.

Ob die Bedienbarkeit eines zyklischen Identitätsautomaten im allgemeinen Fall entscheidbar ist, ist weiterhin offen. Ein Algorithmus zum Synthetisieren eines Partners eines zyklischen Identitätsautomaten konnte in dieser Arbeit nicht gefunden werden.

Wir konnten jedoch zeigen, dass es einen bedienbaren zyklischen Identitätsautomaten gibt, dessen Partner nicht als endlicher Identitätsautomat darstellbar sind. Ein Partner eines Serviceautomaten muss im allgemeinen alle Nachrichten speichern können, die er vom Serviceautomaten erhält. Für zyklische Serviceautomaten können Nachrichtensequenzen beliebig lang werden. Ein Identitätsautomat benötigt deswegen unendlich viele Speicherplätze und ist infolgedessen nicht endlich darstellbar.

Die Anzahl der Nachrichten, die ein azyklischer Serviceautomat verarbeiten kann, ist dagegen stets endlich. Daher muss ein Partner eines azyklischen Serviceautomaten höchstens endlich viele Nachrichten speichern können.

Die Erkenntnis, dass ein Partner jede Nachricht speichern können muss, bestimmt maßgeblich die Art und die Funktionsweise des Partners, den der Partnersynthesealgorithmus, mit dem wir die Bedienbarkeit von azyklischen Formelautomaten entscheiden, konstruiert. Der vom Algorithmus synthetisierte azyklische Partner speichert jede Nachricht, die er erhält, in einem eigenen Speicherplatz solange, bis er einen Endzustand erreicht.

Transformationsregeln Wir haben Transformationsregeln angegeben, welche die Eigenschaft eines Serviceautomaten, Partner eines anderen Serviceautomaten zu sein, erhalten. Diese können verwendet werden, um die Größe eines Partner zu reduzieren.

Wir haben ein Kriterium vorgestellt, um *redundante* Speicherplätze eines azyklischen Partners zu identifizieren, d. h. Speicherplätze, deren Inhalt keine Bedeutung für die weitere Interaktion der Serviceautomaten hat. Solche Speicherplätze können entfernt werden. Der Partner ist nach dem Entfernen des Speicherplatzes wieder ein Partner.

Es wurde eine Regel zum Verschmelzen von Zuständen eines Partners eines azyklischen Serviceautomaten vorgestellt. Der Partner ist nach dem Ausführen der Verschmelzungsoperation wieder ein Partner. Die Verschmelzungsregel haben wir verallgemeinert auf Knoten eines Identitätsautomaten. Das Verschmelzen zweier Knoten entspricht dem Verschmelzen unendlich vieler Zustände.

Offene Probleme Ein offenes Problem ist die Synthese von Partnern zyklischer Serviceautomaten. Im allgemeinen ist die Existenz eines Partners eines zyklischen Serviceautomaten unentscheidbar. Es könnte jedoch eine Klasse Identitätsautomaten existieren, für die Bedienbarkeit im zyklischen Fall entscheidbar ist. Da ein bedienbarer Identitätsautomat nicht immer auch einen endlich darstellbaren Partner hat, muss zunächst eine Klasse gefunden werden, in der jeder bedienbare Serviceautomat auch mindestens einen Partner mit endlich vielen Knoten hat.

Wir vermuteten, dass für einen zyklischen Identitätsautomaten, der einen endlich darstellbaren Partner hat, ein solcher Partner auch algorithmisch konstruiert werden kann. Wir glauben, dass die Identifizierung und Entfernung von Speicherplätzen, welche den Inhalt nicht mehr benötigter Nachrichten speichern, ein zentraler Bestandteil eines Synthesealgorithmus für Partner zyklischer Serviceautomaten sein wird.

Durch Verschmelzen von Knoten und Entfernen redundanter Variablen könnte es unter bestimmten Voraussetzungen möglich sein, einen Partner mit unendlich vielen Knoten und unbeschränkt vielen Speicherplätzen zu einem Partner mit endlich vielen Knoten und endlich vielen Speicherplätzen zu reduzieren. Mit einem geeigneten Algorithmus könnte es möglich sein, zunächst einen baumförmigen Partner mit unbeschränkter Tiefe (bzw. einen endlichen Präfix von diesem) zu konstruieren, der anschließend zu einem zyklischen Partner zusammengefaltet wird. Die Ausarbeitung eines solchen Algorithmus bleibt eine Aufgabe zukünftiger Arbeiten.

Der Ansatz zur Synthese von Adaptern muss ebenfalls verfeinert werden. Ein in praktischen Anwendungsfällen sinnvoll einsetzbarer Adapter muss neben der Einhaltung des Kompatibilitätskriteriums die Einhaltung weiterer Nebenbedingungen garantieren. Die

Fähigkeit des Adapter, Nachrichten zu erzeugen oder umzuwandeln müssen beschränkt werden können auf solche, die im gegebenen Anwendungsfall sinnvoll und möglich sind. Mit dem in der Arbeit vorgestellten Partnersynthesealgorithmus für Formelautomaten ist es gegenwärtig nicht möglich, Nebenbedingungen zu spezifizieren und zu berücksichtigen.

Literaturverzeichnis

[1] Aalst, Wil M. P. van der, Arjan J. Mooij, Christian Stahl und Karsten Wolf: *Service Interaction: Patterns, Formalization, and Analysis.* In: Bernardo, Marco, Luca Padovani und Gianluigi Zavattaro (Herausgeber): *SFM*, Band 5569 der Reihe *Lecture Notes in Computer Science*, Seiten 42–88. Springer, 2009. 68

[2] Abid, Chiheb Ameur und Belhassen Zouari: *Supervisory Control and High-level Petri nets*, Kapitel 14, Seiten 281–306. INTECH, Februar 2010. 123

[3] Alpern, Bowen, Mark N. Wegman und F. Kenneth Zadeck: *Detecting Equality of Variables in Programs.* In: Ferrante, Jeanne und P. Mager (Herausgeber): *POPL*, Seiten 1–11. ACM Press, 1988. 59

[4] Awad, Ahmed, Gero Decker und Niels Lohmann: *Diagnosing and Repairing Data Anomalies in Process Models.* In: Rinderle-Ma, Stefanie, Shazia Sadiq und Frank Leymann (Herausgeber): *Business Process Management Workshops, BPM 2009*, Band 43 der Reihe *Lecture Notes in Business Information Processing*, Seiten 5–16. Springer-Verlag, März 2010. 69

[5] Badouel, Eric und Philippe Darondeau: *Theory of Regions.* In: Reisig, Wolfgang und Grzegorz Rozenberg (Herausgeber): *Petri Nets*, Band 1491 der Reihe *Lecture Notes in Computer Science*, Seiten 529–586. Springer, 1996. 123

[6] Baldan, Paolo, Andrea Corradini, Hartmut Ehrig und Reiko Heckel: *Compositional Modeling of Reactive Systems Using Open Nets.* In: Larsen, Kim Guldstrand und Mogens Nielsen (Herausgeber): *CONCUR*, Band 2154 der Reihe *Lecture Notes in Computer Science*, Seiten 502–518. Springer, 2001. 3, 25

[7] Basu, Samik, Madhavan Mukund, C. R. Ramakrishnan, I. V. Ramakrishnan und Rakesh M. Verma: *Local and Symbolic Bisimulation Using Tabled Constraint Logic Programming.* In: Codognet, Philippe (Herausgeber): *ICLP*, Band 2237 der Reihe *Lecture Notes in Computer Science*, Seiten 166–180. Springer, 2001. 35

[8] Belkhir, Walid, Yannick Chevalier und Michaël Rusinowitch: *Fresh-Variable Automata for Service Composition.* CoRR, abs/1302.4205, 2013. 35

[9] Belkhir, Walid, Yannick Chevalier und Michaël Rusinowitch: *Guarded Variable Automata over Infinite Alphabets.* CoRR, abs/1304.6297, 2013. 35

[10] Bergstra, J. A.: *Handbook of Process Algebra.* Elsevier Science Inc., New York, NY, USA, 2001. 3

[11] Bojanczyk, Mikolaj, Anca Muscholl, Thomas Schwentick, Luc Segoufin und Claire David: *Two-Variable Logic on Words with Data.* In: *LICS*, Seiten 7–16. IEEE Computer Society, 2006. 35

[12] Bolognesi, Tommaso und Ed Brinksma: *Introduction to the ISO Specification Language LOTOS.* Computer Networks, 14:25–59, 1987. 3

[13] Bryant, Randal E.: *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Trans. Computers, 35(8):677–691, 1986. 4, 35, 68, 123

[14] Chen, Yi Liang und Feng Lin: *Modeling of discrete event systems using finite state machines with parameters.* In: *Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on*, Seiten 941–946. IEEE, 2000. 35

[15] Chiola, Giovanni, Claude Dutheillet, Giuliana Franceschinis und Serge Haddad: *On Well-Formed Coloured Nets and their Symbolic Reachability Graph.* In: Jensen, Kurt und Grzegorz Rozenberg (Herausgeber): *High-Level Petri Nets – Theory and Application*, Seiten 373–396. Springer, 1991. 36, 102, 123

[16] Cooper, D. C.: *Theorem Proving in Arithmetic without Multiplication.* In: *Machine Intelligence*, Band 7, Seiten 91–99, New York, 1972. American Elsevier. 84

[17] Cousot, Patrick und Radhia Cousot: *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* In: Graham, Robert M., Michael A. Harrison und Ravi Sethi (Herausgeber): *POPL*, Seiten 238–252. ACM, 1977. 122, 124

[18] Couvreur, Jean Michel, Emmanuelle Encrenaz, Emmanuel Paviot-Adet, Denis Poitrenaud und Pierre André Wacrenier: *Data Decision Diagrams for Petri Net Analysis.* In: Esparza, Javier und Charles Lakos (Herausgeber): *ICATPN*, Band 2360 der Reihe *Lecture Notes in Computer Science*, Seiten 101–120. Springer, 2002. 4, 123

[19] Dijkstra, Edsger W.: *A Discipline of Programming.* Prentice Hall, Inc., October 1976. 87

[20] Fahland, Dirk: *Oclets - scenario-based modeling with Petri nets.* In: Franceschinis, Giuliana und Karsten Wolf (Herausgeber): *Proceedings of the 30th International Conference on Petri Nets and Other Models Of Concurrency, 22-26 May 2009*, Band 5606 der Reihe *Lecture Notes in Computer Science*, Seiten 223–242, Paris, France, Juni 2009. Springer-Verlag. (revised version). 3

[21] Fahland, Dirk und Christian Gierds: *Analyzing and Completing Middleware Designs for Enterprise Integration Using Coloured Petri Nets.* In: Salinesi, Camille, Moira C. Norrie und Oscar Pastor (Herausgeber): *CAiSE*, Band 7908 der Reihe *Lecture Notes in Computer Science*, Seiten 400–416. Springer, 2013. 100

[22] Fahland, Dirk und Robert Prüfer: *Data and Abstraction for Scenario-Based Modeling with Petri Nets.* In: *Petri Nets*, Seiten 168–187, 2012. 3

[23] Findlow, Greg: *Obtaining Deadlock-Preserving Skeletons for Coloured Nets.* In: Jensen, Kurt (Herausgeber): *Application and Theory of Petri Nets*, Band 616 der Reihe *Lecture Notes in Computer Science*, Seiten 173–192. Springer, 1992. 68

[24] Fischer, Michael J. und Michael O. Rabin: *Super-exponential complexity of Presburger arithmetic.* Complexity of Comput., Proc. Symp. appl. Math., New York City 1973, 27-41 (1974)., 1974. 84

[25] Garg, Vijay K und Ratnesh Kumar: *A state-variable approach for controlling discrete event systems with infinite states.* In: *American Control Conference, 1992*, Seiten 2809–2813. IEEE, 1992. 122

[26] Gierds, Christian: *Finding Cost-Efficient Adapters.* In: Lohmann, Niels und Karsten Wolf (Herausgeber): *AWPN*, Band 380 der Reihe *CEUR Workshop Proceedings*, Seiten 37–42. CEUR-WS.org, 2008. 72

[27] Gierds, Christian, Arjan J. Mooij und Karsten Wolf: *Reducing Adapter Synthesis to Controller Synthesis.* IEEE T. Services Computing, 5(1):72–85, 2012. 72, 100

[28] Gierds, Christian und Jan Sürmeli: *Estimating costs of a service.* In: Gierds, Christian und Jan Sürmeli (Herausgeber): *Proceedings of the 2nd Central-European Workshop on Services and their Composition, ZEUS 2010, Berlin, Germany, February 25–26, 2010*, Band 563 der Reihe *CEUR Workshop Proceedings*, Seiten 121–128. CEUR-WS.org, 2010. 67

[29] Grumberg, Orna, Orna Kupferman und Sarai Sheinvald: *Variable Automata over Infinite Alphabets.* In: Dediu, Adrian Horia, Henning Fernau und Carlos Martín-Vide (Herausgeber): *LATA*, Band 6031 der Reihe *Lecture Notes in Computer Science*, Seiten 561–572. Springer, 2010. 35

[30] Hameurlain, Nabil und Christophe Sibertin-Blanc: *Finite Symbolic Reachability Graphs for High-Level Petri Nets.* In: *APSEC*, Seiten 150–159. IEEE Computer Society, 1997. 36, 104

[31] Heckel, Reiko: *Open Petri Nets as Semantic Model for Workflow Integration.* In: Ehrig, Hartmut, Wolfgang Reisig, Grzegorz Rozenberg und Herbert Weber (Herausgeber): *Petri Net Technology for Communication-Based Systems*, Band 2472 der Reihe *Lecture Notes in Computer Science*, Seiten 281–294. Springer, 2003. 25

[32] Hee, Kees M. van, Natalia Sidorova, Marc Voorhoeve und Jan Martijn E. M. van der Werf: *Generation of Database Transactions with Petri Nets.* Fundam. Inform., 93(1-3):171–184, 2009. 36

[33] Heinze, Thomas S., Wolfram Amme und Simon Moser: *Effiziente Abschätzung von Datenflussfehlern in strukturierten Geschäftsprozessen.* In: Eichhorn, Daniel, Agnes

Koschmider und Huayu Zhang (Herausgeber): *ZEUS*, Band 705 der Reihe *CEUR Workshop Proceedings*, Seiten 73–80. CEUR-WS.org, 2011. 69

[34] Hennessy, Matthew und Huimin Lin: *Symbolic Bisimulations.* Theor. Comput. Sci., 138(2):353–389, 1995. 35

[35] Hoare, C. A. R.: *Communicating Sequential Processes.* Prentice-Hall, 1985. 3

[36] Holloway, Lawrence E., Bruce H. Krogh und Alessandro Giua: *A Survey of Petri Net Methods for Controlled Discrete Event Systems.* Discrete Event Dynamic Systems, 7(2):151–190, 1997. 123

[37] Jensen, Kurt: *Condensed State Spaces for Symmetrical Coloured Petri Nets.* Formal Methods in System Design, 9(1/2):7–40, 1996. 36, 104

[38] Jensen, Kurt und Lars Michael Kristensen: *Coloured Petri Nets - Modelling and Validation of Concurrent Systems.* Springer, 2009. 4, 12, 35

[39] Junttila, Tommi A.: *New Canonical Representative Marking Algorithms for Place/Transition-Nets.* In: Cortadella, Jordi und Wolfgang Reisig (Herausgeber): *ICATPN*, Band 3099 der Reihe *Lecture Notes in Computer Science*, Seiten 258–277. Springer, 2004. 36, 104

[40] Kalyon, Gabriel, Tristan Le Gall, Hervé Marchand und Thierry Massart: *Symbolic Supervisory Control of Infinite Transition Systems Under Partial Observation Using Abstract Interpretation.* Discrete Event Dynamic Systems, 22(2):121–161, 2012. 122

[41] Kalyon, Gabriel, Tristan Le Gall, Hervé Marchand und Thierry Massart: *Control of Infinite Symbolic Transition Systems under Partial Observation.* In: *European Control Conference*, Budapest Hungary, August 2009. 122

[42] Kaminski, Michael und Nissim Francez: *Finite-Memory Automata.* Theor. Comput. Sci., 134(2):329–363, 1994. 35

[43] Kaschner, Kathrin, Peter Massuthe und Karsten Wolf: *Symbolic Representation of Operating Guidelines for Services.* Petri Net Newsletter, 72:21–28, April 2007. 68

[44] Klaedtke, Felix: *Bounds on the automata size for Presburger arithmetic.* ACM Trans. Comput. Log., 9(2), 2008. 84

[45] Klai, Kais, Samir Tata und Jörg Desel: *Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes.* Data Knowl. Eng., 70(5):467–482, 2011. 68

[46] Kumar, Ratnesh und Vijay K Garg: *State Avoidance Control for Infinite State Systems using Assignment Program Model.* In: *Discrete Event Dynamical Systems: Theory and Applications.* Citeseer, 1995. 122

[47] Kumar, Ratnesh und Vijay K. Garg: *On computation of state avoidance control for infinite state systems in assignment program framework*. IEEE T. Automation Science and Engineering, 2(1):87–91, 2005. 122

[48] Kupferman, Orna und Moshe Y. Vardi: *Modular Model Checking*. In: Roever, Willem P. de, Hans Langmaack und Amir Pnueli (Herausgeber): *COMPOS*, Band 1536 der Reihe *Lecture Notes in Computer Science*, Seiten 381–401. Springer, 1997. 67

[49] Lamport, Leslie: *The Temporal Logic of Actions*, 1993. 40, 67

[50] Lazic, Ranko, Tom Newcomb, Joël Ouaknine, A. W. Roscoe und James Worrell: *Nets with Tokens which Carry Data*. Fundam. Inform., 88(3):251–274, 2008. 36

[51] Lin, Huimin: *Symbolic Transition Graph with Assignment*. In: Montanari, Ugo und Vladimiro Sassone (Herausgeber): *CONCUR*, Band 1119 der Reihe *Lecture Notes in Computer Science*, Seiten 50–65. Springer, 1996. 35

[52] Lohmann, Niels: *A Feature-Complete Petri Net Semantics for WS-BPEL 2.0 and its Compiler BPEL2oWFN*. Informatik-Berichte 212, Humboldt-Universität zu Berlin, August 2007. 3

[53] Lohmann, Niels: *Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance*. In: *BPM 2008*, LNCS. Springer-Verlag, September 2008. 72

[54] Lohmann, Niels: *Why Does My Service Have No Partners?* In: Bruni, Roberto und Karsten Wolf (Herausgeber): *WS-FM*, Band 5387 der Reihe *Lecture Notes in Computer Science*, Seiten 191–206. Springer, 2008. 72

[55] Lohmann, Niels: *Communication models for services*. In: Gierds, Christian und Jan Sürmeli (Herausgeber): *ZEUS*, Band 563 der Reihe *CEUR Workshop Proceedings*, Seiten 9–16. CEUR-WS.org, 2010. 25

[56] Lohmann, Niels: *Correctness of services and their composition*. Dissertation, Universität Rostock / Technische Universiteit Eindhoven, Rostock, Germany / Eindhoven, The Netherlands, September 2010. 40, 68

[57] Lohmann, Niels, Peter Massuthe und Karsten Wolf: *Operating Guidelines for Finite-State Services*. In: Kleijn, Jetty und Alex Yakovlev (Herausgeber): *28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings*, Band 4546 der Reihe *Lecture Notes in Computer Science*, Seiten 321–341. Springer-Verlag, 2007. 3, 25, 27, 37, 41, 68, 78, 79

[58] Lohmann, Niels und Karsten Wolf: *Data under control*. In: *Proceedings of the 18th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2011), Hagen, Germany, September 29-30, 2011*, September 2011. 122

[59] Maréchal, Olivier, Pascal Poizat und Jean Claude Royer: *Checking asynchronously communicating components using symbolic transition systems.* In: *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, Seiten 1502–1519. Springer, 2004. 35

[60] Markovski, Jasen: *Communicating Processes with Data for Supervisory Coordination.* In: Kokash, Natallia und António Ravara (Herausgeber): *FOCLASA*, Band 91 der Reihe *EPTCS*, Seiten 97–111, 2012. 123

[61] Markovski, Jasen: *Controllability for Nondeterministic Finite Automata with Variables.* In: Cordeiro, José, David A. Marca und Marten van Sinderen (Herausgeber): *ICSOFT*, Seiten 438–446. SciTePress, 2013. 35

[62] Massuthe, Peter, Alexander Serebrenik, Natalia Sidorova und Karsten Wolf: *Can I find a Partner? Undecidability of Partner Existence for Open Nets.* Information Processing Letters, 108(6):374–378, November 2008. 48

[63] Meda, Hema S., Anup K. Sen und Amitava Bagchi: *On Detecting Data Flow Errors in Workflows.* J. Data and Information Quality, 2(1), 2010. 69

[64] Milner, Robin: *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989. 3

[65] Milner, Robin: *Communicating and mobile systems - the Pi-calculus.* Cambridge University Press, 1999. 15

[66] Minsky, Marvin L.: *Computation: finite and infinite machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967. 48

[67] Monk, J. Donald: *Mathematical Logic.* Springer-Verlag, 1976. 81, 84

[68] Neven, Frank, Thomas Schwentick und Victor Vianu: *Finite state machines for strings over infinite alphabets.* ACM Transactions on Computational Logic (TOCL), 5(3):403–435, 2004. 35

[69] Ouedraogo, Lucien, Ratnesh Kumar, Robi Malik und Knut Akesson: *Nonblocking and safe control of discrete-event systems modeled as extended finite automata.* Automation Science and Engineering, IEEE Transactions on, 8(3):560–569, 2011. 123

[70] Parnjai, Jarungjit: *Behavioral Service Substitution: Analysis and Synthesis.* Dissertation, Humboldt-Universität zu Berlin, 2013. 68

[71] Pathak, Jyotishman, Samik Basu und Vasant Honavar: *Modeling web service composition using symbolic transition systems.* In: *AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing*, Seiten 44–51, 2006. 35

[72] Pnueli, Amir: *The Temporal Logic of Programs.* In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, Seiten 46–57, Washington, DC, USA, 1977. IEEE Computer Society. 40, 67

[73] Pommereau, Franck, Raymond R. Devillers und Hanna Klaudel: *Efficient Reachability Graph Representation of Petri Nets With Unbounded Counters.* Electr. Notes Theor. Comput. Sci., 239:119–129, 2009. 36

[74] Presburger, M.: *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt.* In: *Comptes Rendus du Premier Congrès des Mathématicienes des Pays Slaves*, Seiten 92–101, 395, Warsaw, 1929. 84

[75] Ramadge, P. J. G. und W. M. Wonham: *The control of discrete event systems.* Proceedings of the IEEE, 77:81–98, 1989. 41

[76] Ramadge, P.J. und W.M. Wonham: *Supervisory control of a class of discrete event processes.* In: Bensoussan, A. und J.L. Lions (Herausgeber): *Analysis and Optimization of Systems*, Band 63 der Reihe *Lecture Notes in Control and Information Sciences*, Seiten 475–498. Springer Berlin Heidelberg, 1984. 122

[77] Ratzer, Anne V., Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen und Kurt Jensen: *CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets.* In: Aalst, Wil M. P. van der und Eike Best (Herausgeber): *ICATPN*, Band 2679 der Reihe *Lecture Notes in Computer Science*, Seiten 450–462. Springer, 2003. 4

[78] Reisig, Wolfgang: *Petrinetze, Eine Einführung.* Springer, 1982. Mit Übersetzungen ins Chinesische, Englische, Italienische, Japanische und Polnische. 3

[79] Reisig, Wolfgang: *On the Expressive Power of Petri Net Schemata.* In: *ICATPN 2005, Miami, USA. Proceedings*, Band 3536 der Reihe *Lecture Notes in Computer Science*, Seiten 349–364. Springer Verlag, Mai 2005. 4

[80] Salaün, Gwen, Lucas Bordeaux und Marco Schaerf: *Describing and Reasoning on Web Services using Process Algebra.* In: *ICWS*, Seiten 43–. IEEE Computer Society, 2004. 3

[81] Schmidt, Karsten: *Parameterized Reachability Trees for Algebraic Petri Nets.* In: Michelis, Giorgio De und Michel Diaz (Herausgeber): *Application and Theory of Petri Nets*, Band 935 der Reihe *Lecture Notes in Computer Science*, Seiten 392–411. Springer, 1995. 36, 122

[82] Schmidt, Karsten: *Symbolische Analysemethoden für algebraische Petri-Netze.* Dissertation, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 1996. 36

[83] Schmidt, Karsten: *Controllability of Open Workflow Nets.* In: Desel, Jörg und Ulrich Frank (Herausgeber): *EMISA*, Band 75 der Reihe *LNI*, Seiten 236–249. GI, 2005. 41

[84] Sköldstam, Markus, Knut Åkesson und Martin Fabian: *Modeling of Discrete Event Systems using Finite Automata With Variables*. In: *Proceedings of the 46th IEEE Conference on Decision and Control*, 2007. 10, 35, 123

[85] Sköldstam, Markus, Knut Åkesson und Martin Fabian: *Supervisory control applied to automata extended with variables*. Technischer Bericht, Chalmers University of Technology, 2007. 123

[86] Stahl, Christian, Peter Massuthe und Jan Bretschneider: *Deciding Substitutability of Services with Operating Guidelines*. Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems, 2(5460):172–191, März 2009. 68

[87] Stahl, Christian und Walter Vogler: *A Trace-Based View on Operating Guidelines*. In: Hofmann, Martin (Herausgeber): *FOSSACS*, Band 6604 der Reihe *Lecture Notes in Computer Science*, Seiten 411–425. Springer, 2011. 68

[88] Sun, Sherry X., J. Leon Zhao, Jay F. Nunamaker und Olivia R. Liu Sheng: *Formulating the Data-Flow Perspective for Business Process Management*. Information Systems Research, 17(4):374–391, 2006. 69

[89] Sürmeli, Jan und Marvin Triebel: *Synthesizing Cost-Minimal Partners for Services*. In: Basu, Samik, Cesare Pautasso, Liang Zhang und Xiang Fu (Herausgeber): *Service-Oriented Computing*, Band 8274 der Reihe *Lecture Notes in Computer Science*, Seiten 584–591. Springer Berlin Heidelberg, 2013. 67

[90] Thierry-Mieg, Yann, Jean Michel Ilié und Denis Poitrenaud: *A Symbolic Symbolic State Space Representation*. In: Frutos-Escrig, David de und Manuel Núñez (Herausgeber): *FORTE*, Band 3235 der Reihe *Lecture Notes in Computer Science*, Seiten 276–291. Springer, 2004. 123

[91] Trčka, Nikola und Natalia Sidorova: *Data-flow anti-patterns: Discovering data-flow errors in workflows*. In: *CAiSE 2009. LNCS 5565*, Seite 425. Springer, 2009. 69

[92] Vautherin, Jacques: *Parallel systems specitications with coloured Petri nets and algebraic specifications*. In: Rozenberg, Grzegorz (Herausgeber): *European Workshop on Applications and Theory of Petri Nets*, Band 266 der Reihe *Lecture Notes in Computer Science*, Seiten 293–308. Springer, 1986. 4

[93] Vogler, Walter, Christian Stahl und Richard Müller: *A Trace-Based Semantics for Responsiveness*. In: Brandt, Jens und Keijo Heljanko (Herausgeber): *Proceedings of the 12th International Conference on Application of Concurrency to System Design, ACSD 2012, Hamburg, Germany, June 27-29, 2012*, Seiten 42–51. IEEE, 2012. 67, 68

[94] Wagner, Christoph: *Partner datenverarbeitender Services*. In: *AWPN 2010*. CEUR-WS.org, Oktober 2010. 54

[95] Weinberg, Daniela: *Deciding Service Substitution - Termination guaranteed.* Dissertation, Universität Rostock, 2012. 40, 67, 68

[96] Wolf, Karsten: *Does my service have partners?* LNCS ToPNoC, 5460(II):152–171, März 2009. Special Issue on Concurrency in Process-Aware Information Systems. 37, 41, 45, 48, 74, 78, 122, 124

[97] Wolfang Reisig: *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien.* Leitfäden der Informatik. Vieweg+Teubner, July15 July 2010. 248 pages; ISBN 978-3-8348-1290-2. 35

[98] Zhou, Changyan und Ratnesh Kumar: *Finite Bisimulation of Reactive Untimed Infinite State Systems Modeled as Automata With Variables.* IEEE T. Automation Science and Engineering, 10(1):160–170, 2013. 35

Index

Erklärung

Hiermit erkläre ich, dass

- ich die vorliegende Dissertationsschrift „Partner datenverarbeitender Services“ selbständig und ohne unerlaubte Hilfe angefertigt sowie nur die angegebene Literatur verwendet habe,
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder ich einen solchen besitze und
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin (veröffentlicht im Amtlichen Mitteilungsblatt der Humboldt-Universität Nr. 34/2006) bekannt ist.

Christoph Wagner
Berlin, den 3. Juni 2014